Modeling and analyzing Helsinki's traffic network using a microscopic simulator

Ivan Pallarès

School of Electrical Engineering

Final Project
Espoo 15.7.2019

Supervisor and advisor

Prof. Themistoklis Charalambous

Aalto University
School of Electrical
Engineering

Aalto University
School of Electrical
Engineering

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Ivan Pallarès

Title Modeling and analyzing Helsinki's traffic network using a microscopic simulator

Master's Programme Automation and Electrical Engineering

Major Automation **Code of major** -

Supervisor and advisor Prof. Themistoklis Charalambous

Date 15.7.2019 **Number of pages** 34+26 **Language** English

Abstract

In today's urban transportation networks, traffic is regulated by traffic signals which control vehicle flows in road intersections and other road spaces. Generally, controllers follow a fixed-cycle schedule which is previously designed. Under heavy or changing conditions, these standard methods may prove insufficient, and it is where adaptive traffic signaling at intersections can be the solution.

This thesis examines a part of Helsinki's city center network to verify theories of traffic flow and tests the performance of the Max-Weight algorithm under different scenarios. Traffic simulations are accomplished using PTV Vissim, which is a microscopic traffic simulator, whereas MATLAB R2018b executes the algorithms, the filter, and plots the results.

The analysis led to the following conclusions: the Max-Weight algorithm control outperforms the traditional fixed and cyclic control under rush hour situation in Helsinki. Nevertheless, under noisy measurements, the tested filtered Max-Weight algorithm control did not make such a difference, compared with the non-filtered noisy control. Further investigations are proposed referring to the particle filter.

Keywords Traffic Networks, Traffic Control, Modeling, Max-Weight algorithm, Back-Pressure algorithm, Particle filter

Preface

I want to thank Professor Themistoklis Charalambous for his good guidance.

Otaniemi, 27.6.2019

Ivan Pallarès

Contents

Symbols and abbreviations

Abbreviations

BP	Back-Pressure
MW	Max-Weight
F	Phase F: free-flow phase
S	Phase S: synchronized phase
J	Phase J: wide moving jam
OD	Origin-Destination

1 Introduction

Traffic condition is gaining interest in major cities since it has a direct impact on transportation, the environment, and human health. Over the past century, urban settlements are getting increasingly more inhabited, whereas urban infrastructures cannot grow or improve at the same level. This uneven growth is inevitably leading to an increase in traffic density, and it is forcing cities to enhance their traffic control.

In today's urban transportation networks, traffic is regulated by traffic signals which control vehicle flows in road intersections and other road spaces. Generally, controllers follow a fixed-cycle schedule which is previously designed. Under heavy or changing conditions, these standard methods may prove insufficient, and it is where adaptive traffic signaling at intersections can be the solution.

Many different algorithms are used in this field. However, an optimal approach is the Max-Weight (or back-pressure) algorithm, which is a method for dynamically routing traffic over a queuing network that leads to maximum network throughput. Nevertheless, the performance of the algorithm could be affected by noisy measurements, which is why the use of a particle filter might be interesting to test in the network.

This thesis examines a part of Helsinki's city center network to verify theories and models of traffic flow. Moreover, this project tests the performance of the Max-Weight algorithm and tries the particle filter in different scenarios. Traffic simulations are accomplished using PTV Vissim, which is a microscopic traffic simulator, whereas MATLAB R2018b executes the algorithms, the filter, and plots all the results.

The thesis is organized as follows. In section 2, the theoretical part of the project, as well as the methods and the algorithms used, are explained. In section 3, the Vissim model, the execution of the simulation, and all the used material are described. Then, the results are presented and analyzed in section 4. Finally, in section 5, conclusions and future directions are drawn.

2 Background

2.1 Microscopic and macroscopic traffic flow modeling

Traffic models are used for various applications. For instance, they can be applied when adjusting the infrastructure or trying to solve the current known traffic problems. In general, they are useful for analyzing traffic flow.

There are different types of traffic models: microscopic, mesoscopic, and macroscopic models. The difference between the models is their level of detail regarding the network. In this thesis, we focus on microscopic and macroscopic models.

2.1.1 Microscopic traffic simulation

Microscopic traffic models are an essential tool for traffic analysis. They define the details of traffic flow and the interaction taking place within it. In short, they simulate single vehicle-driver units. The position or the velocity of the single vehicles are variables of the models.

These models can be classified into two main categories: cell automata models, which are discrete in time and space, and car-following models (also known as continuous models), which are continuous in time [3]. In the cell automata models, the network is formed by vehicles that move between cells in discrete steps. Usually, these models are simpler and faster to simulate but result in less accurate and realistic measurements. On the other hand, more computational capacities are required by car-following models due to their complexity, but more realistic results are obtained.

Moreover, car-following models follow two main categories of theories. The first category is known as follow-the-leader theories, and it is based on the idea that each vehicle has to maintain a safe distance with the other vehicles. The second category assumes that the vehicle moves at a desired (and legal) velocity of the driver. The maximum velocity decreases if the traffic increases.

The primary traffic model of the software used in this thesis, PTV VISSIM, is based on a car-following model that considers different aspects of the drivers. Rainer Wiedemann developed the model in 1974 at Karlsruhe University [1].

2.1.2 Macroscopic traffic simulation

In contrast, macroscopic traffic modeling considers vehicles as particles flowing in a tube (the road), similar to liquid or gas models. Traffic is evaluated thanks to three main variables:

- The flow rate $q(x,t)$: number of vehicles passing a fixed point x per unit of time.
- The flow density $\rho(x,t)$: number of vehicles per unit of length.
- The vehicle velocity $v(x,t)$: flow velocity in a fixed position per unit of time.

The variables are related by the flow-density relation:

$$q(x,t) = \rho(x,t)v(x,t)$$

Macroscopic variables can be measured by integrating microscopic traffic flow models. Flow rate is obtained by counting vehicles passing a fixed point for a period of time. Velocity is calculated as the mean of the vehicle's velocity that passed the fixed point for a period of time. Density is determined thanks to the flow-density relation. In this thesis, the microscopic model is used to simulate all the network and the traffic, whereas the macroscopic model is applied in order to perform part of the analysis [2][3].

2.2 Three-phase model

2.2.1 Fundamental diagram of traffic flow

The fundamental diagram of traffic flow is a diagram that relates the traffic flow (vehicles/time unit) and traffic density (vehicles/distance unit). It is based on the Macroscopic traffic model, which uses the flow-density relation. Thanks to this diagram, studies about the behavior and the capability of the roads can be carried. It is the primary tool for graphically presenting information in the study of traffic flow[4].

The diagram is composed of two branches: the free-flowing branch, which is the line with a positive slope, and the congested branch, which has a negative slope. The traffic flow increases with the increase of the traffic density until it reaches the maximum point of traffic flow ρ_c, which is called the critical density. From this point on, the higher the density on the congested branch, the lower the traffic flow: the congestion grows until the flow is completely stopped. Fig. 1 shows a simplified version of the fundamental diagram of traffic flow [5].

2.2.2 Three-phase theory

The three-phase theory offers a qualitative analysis of real traffic, and it is based on the same principle as the fundamental diagram of traffic flow. However, the three-phase theory divides the congested branch into two other parts, synchronized flow S and wide moving jam J. The third phase is the so-called free flow phase.

In the same way, in free flow (phase F), the flow rate is approximately proportional to the density: an increase in the traffic density results in a gain of flow. In the synchronized phase (phase S), the average speed of vehicles drops, whereas the flow rate is maintained. This is because the product of the speed and the density remains the same, and the density increases. The synchronization of the speed of the vehicles in different lanes names the phase. Finally, the wide moving jam stage (phase J) occurs when the traffic flow and speed decrease significantly, and the density achieves its maximum. The wide moving jam can only happen through a synchronized flow.

The transition from F to S occurs after the density reaches the critical point ρ_c. In a real situation, random fluctuations of the traffic, due to vehicle deceleration,

lane changing, or erratic behavior of drivers can cause the F to S transition before the critical point. The nearer of the critical point, the higher is the probability of phase change. Unlike the free flow phase, the synchronized phase is represented as an area in the flow-density diagram because of the nature of the fluctuations.

The transition to phase J can only occur when the traffic is in synchronized flow (phase S). Consequently, the flow and speed of vehicles drop. The direct transition from free flow phase is not possible [4].

In this thesis, the three-phase model will be used to analyze the state of the traffic.

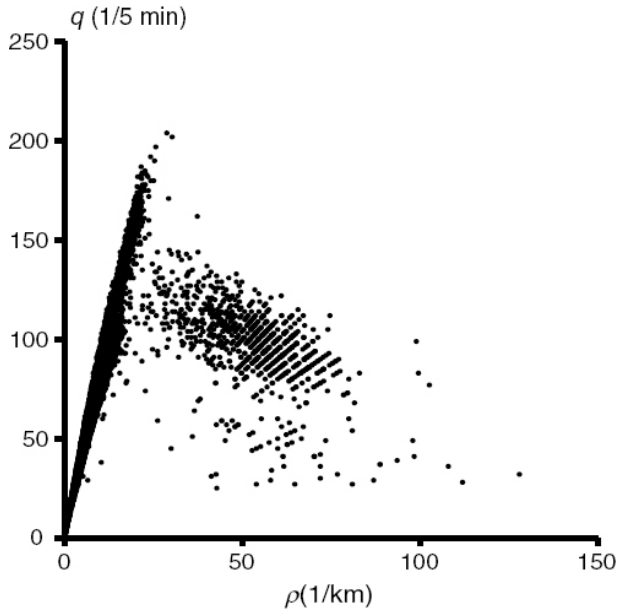

Figure 1: The typical fundamental diagram (the relation between vehicle density and flow rate) from 1 month of data measured at a point on a freeway. The critical density is nearly 25 (vehicles/km). The data were measured by the Japan Highway Public Cooperation [6].

2.3 The Max-Weight algorithm

The max-weight algorithm, also known as the back-pressure algorithm, is a method for dynamically routing traffic over a queueing network that leads to maximum network throughput. It was first proposed in the seminal paper [7] by Tassiulas and Ephremides, and since then, it has been used and enhanced, mostly in communication network applications.

However, in this thesis, the algorithm is applied in transportation systems. This work implements the solution proposed in [8], and it considers decentralized traffic signal control policies. The paper demonstrates that the standard Max-Weight algorithm is optimal even under noisy queue measurements, and the use of filtering improves the performance of the algorithm under the same conditions. In this project, we will test and analyze the algorithm under different circumstances.

2.3.1 System model

The model used for representing the traffic network is a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, also known as standard road network model. A directed graph is a graph that is formed by vertices connected by directed edges. The vertices (or nodes) i, represent the lanes of the roads, which are capable of queuing vehicles, and edges (or links) $\varepsilon_{ij} \in \mathcal{E}$ represent the possible flows or paths from the lane (or node) i to lane j. Lane i is known as the outgoing lane, and lane j is also called incoming lane. Roads can have multiple lanes, and each lane is a different node. Every road intersection or junction is the set of links (vertices) that contains all the possible paths between lanes (edges) that reach the intersection.

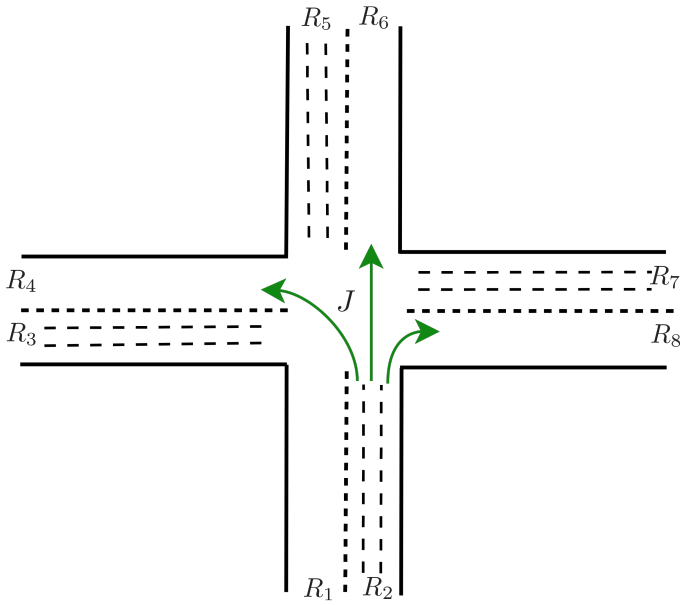

Figure 2: A four-way junction with 8 roads and 12 possible traffic movements [11].

It is assumed that decisions are taken synchronously and that all the controllers at the intersections have the same cycle of length T. Therefore, time is considered as discrete and idle times during switches are not taken into account.

2.3.2 Services rates and traffic phases

During the simulation, the number of vehicles served at time t, from the selected node i to its out-neighbors j is denoted as $S_{ij}(t)$.

Each intersection controlled by a group of traffic lights has a finite number of phases P_l. For each phase $\sigma \in P_l$, a set of non-overlapping links are activated at every slot $t \in N$, and a number of vehicles $S_{ij}(t)$ are transferred by lane i to lanes j. The activation of a concrete phase is decided by the Max-Weight algorithm.

2.3.3 Pressure release and phase choice

The algorithm is designed to make decisions in order to minimize the sum of squares of queue backlogs in the network from one time-slot to the next. It chooses transmission and routing variables to greedily minimize a bound on the drift of a Lyapunov function [9][10]. Specifically, the current queue of a node i at the beginning time slot is denoted as $Q_i(t)$. For each lane i going to lane j, the pressure weight W_{ij} is computed as follows:

$$W_{ij}\Big(Q(t)\Big) = Q_i(t) - Q_j(t)$$

For each phase $\sigma \in P_l$, the junction computes the pressure release

$$w_\sigma\Big(Q(t)\Big) = \sum_{(i,j)\in\sigma} W_{ij}\Big(Q(t)\Big)S_{ij}(t), \forall\sigma \in P_l,$$

and chooses the one with the highest pressure release, i.e.,

$$\sigma^*(t) = \arg\max_{\sigma\in P_l} w_\sigma\Big(Q(t)\Big)$$

In this thesis it is considered that the service rate is the same for all the lanes. In that case, the pressure release can be computed as follows:

$$w_\sigma\Big(Q(t)\Big) = \sum_{(i,j)\in\sigma} W_{ij}\Big(Q(t)\Big), \forall\sigma \in P_l,$$

2.3.4 The Noisy-Max-Weight (or Noisy-Back-Pressure) algorithm

If it is considered that the decisions are made on noisy measurements, then the queue backlog of a node i is noted as $\widetilde{Q}_i(t)$, and $\nu_i(t)$ expresses the noise at lane i at a time t.

$$\widetilde{Q}_i(t) = Q_i(t) + \nu_i(t)$$

Since it is considered that queues are finite and positive, $\nu_i(t)$ is finite for all i. For each lane i going to lane j and for the case where the queue length measurements are noisy, the pressure weight W_{ij} is computed as follows:

$$W_{ij}\Big(\widetilde{Q}(t)\Big) = \widetilde{Q}_i(t) - \widetilde{Q}_j(t)$$

After this point, the pressure release and phase choice is made as in the case mentioned above, using the noisy pressure weight $W_{ij}\Big(\widetilde{Q}(t)\Big)$.

2.4 The Filtered-Max-Weight algorithm

Paper [8] proposes the use of a Bayesian sequential estimator to deal with the noise in the queue measurements. More specifically, it suggests the use of a particle filter, thanks to its ease of implementation and flexibility. During the simulation, the particle filter estimates the queue length for every lane in the network, and then, the max-weight algorithm decides the traffic scheduling from the estimations made by the particle filter.

The proposed filter provides a distributed particle filter via a belief propagation framework. Belief propagation is a message-passing algorithm for performing inference on graphical models. Lanes (or nodes) pass their state or information to their out-neighbors' lanes in order to estimate more precisely their state.

As noted previously, the filtering problem consists of estimating the internal states in a dynamical system, as a traffic network. The objective is to compute states from partial or noisy observations such as noisy cameras measurements, where vehicles can remain occluded during the recognition. This estimation is performed using a set of particles (or samples) that represents the distribution of the given noisy observations.

Belief propagation involves two steps: prediction and correction. First, in the prediction step, the set of particles is obtained by predicting the possible state thanks to the information given by in-neighbors lanes. Then, each particle is assigned a weight that represents the probability of that particle being sampled from the probability density function. The weights are normalized, and particles are resampled (or corrected) following the probability mass function.

Note that each lane (or node) has its own set of weights, unlike the standard particle filter.

Finally, each lane i can determine the most probable queue state as

$$\widehat{Q}_i(t) = \arg\max_{Q_i(t)} p(Q_i(t)|\widetilde{Q}_i^{0:t})$$

And it can be used to compute the pressure release.

More detailed information about the Filtered-Max-Weight algorithm can be found in [8].

3 Research material and methods

This thesis examines a part of Helsinki's city center network to verify theories of traffic flow and to test the performance of the Max-Weight algorithm in different scenarios. Microscopic simulations were carried out with PTV Vissim 11.0, and algorithms calculations and data analysis were performed thanks to MATLAB R2018b.

3.1 City network

3.1.1 Map data

The south part of the Kamppi area, in Helsinki, was chosen to carry out the simulations. This area was selected due to its distribution of the streets, which is homogenous, and comparable to the Eixample neighborhood in Barcelona [18]. This uniformity of the network is a significant advantage since it allows a homogenous distribution of the traffic during the simulation, and it reduces the probability of undesired bottlenecks and deadlocks. Furthermore, Kamppi was chosen because of its high vehicle density during rush hours.

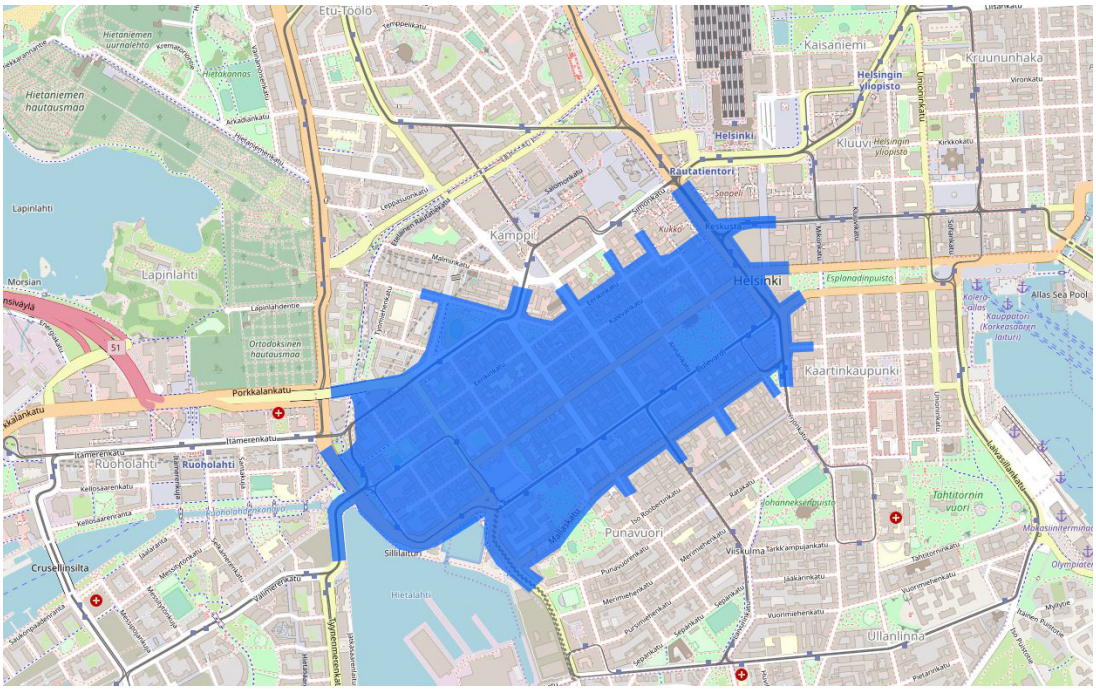

Figure 3: The selected area to carry out the simulations. Kamppi, Helsinki [12].

The map data was imported thanks to OpenStreetMap under the Open Database License [12]. Data was first filtered and adjusted with PTV Visum. Once the desired area was cropped and the suited data preserved (only road data), the network was exported to a file format compatible with PTV Vissim. Additionally, other manual adjustments were performed with PTV Vissim, such as the removal of small and unused road sections, the improvement of road sections, right of priority removal in

conflict areas in junctions, or modification and addition of slow zones in some roads, in order to avoid unrealistic routes, bottlenecks, and deadlocks.

The chosen network is composed of 27 controlled junctions, regulated with two, three or four phases, depending on the intersection. Each phase has a fixed duration of 20 seconds. Figure 4 shows the controlled junctions in the network.

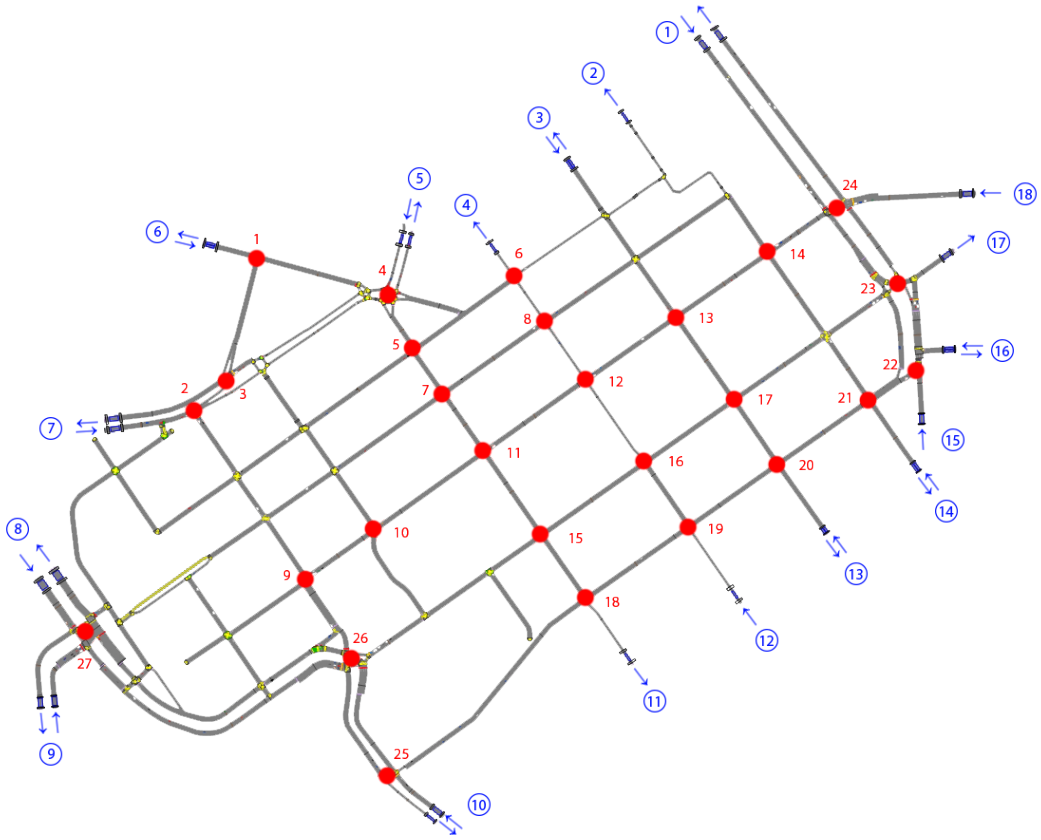

Figure 4: PTV Vissim network. In red, the controlled junctions. In blue, the origins and destinations of the origin-destination matrix.

3.1.2 Traffic demand and routing

The traffic demand was specified by using the dynamic assignment add-on module in PTV Vissim. Instead of designating vehicle inputs on selected links with a given traffic volume, and creating the routes manually, the dynamic assignment employs origin-destination matrices (or OD matrices) [13]. By using these matrices, starting and end points of trips, and the number of trips between these locations have to be specified. The dimension of the OD matrices corresponds to the number of zones used as starting and end points.

The OD matrix was designed to represent the real traffic in this zone during the rush hour, between 8:00 and 8:30. The same matrix was used between 8:30 and 9:00 if a one hour of simulation was required. The matrix was estimated thanks to the traffic data published by the Helsinki Urban Environment Division [14], and

adjusted with real-time data traffic from Google Maps [15]. In order to test the network with different traffic inflows, this matrix was modified by a coefficient before the simulations. In the nominal case (inflow multiplier = 1.0), the inflow rate can reach 7968 vehicles per hour. Figure 4 also shows the origins and destinations of the vehicles in the network.

In the dynamic assignment, PTV Vissim performs a path search, before the path selection (if this option is checked in the dynamic assignment configuration), and then it distributes traffic across all known paths. Additional costs and slow zones were added in required links in order to adjust the path distribution, and obtain a realistic traffic routing.

Finally, it is important to note that there is a difference between the planned traffic, the OD matrix, and the served traffic, in a simulation. Because of the queues located in (or near) starting points or the amount of traffic in the network, the planned quantity of vehicles may not be inserted. This difference will be analyzed in the results section.

3.2 Simulation

3.2.1 Cases and important parameters

Simulations were structured in four main cases. The first case corresponds to the basic situation where signal head controllers switch phases cyclically and fixedly, without decisions. Cases two, three and four employ the Max-Weight algorithm to manage the signal heads. For cases three and four, noise is added to the queue measurements, and two variants are identified for both cases: different scale of noise is added, 10% and 25%. The meaning of the percentage will be explained further on. Finally, the fourth case adopts the particle filter in order to estimate the queue measurements. Table 1 shows the description of each case.

Cases List			
Case Code	Max-Weight Algorithm	Queue Noise Level	Particle Filter
100	No	0%	No
200	Yes	0%	No
310	Yes	10%	No
325	Yes	25%	No
410	Yes	10%	Yes
425	Yes	25%	Yes

Table 1: Cases List

In addition, each case was simulated with different vehicle inflow values, in order to identify its limits and performance and gather an adequate amount of data. In PTV Vissim, this is achieved by multiplying the OD matrix by a rate. Table 2 shows the different inflow multipliers used during the simulations. In total, 42 simulations were launched.

Inflow List							
Inflow Code	050	075	100	125	150	200	250
Inflow Multiplier	0.5	0.75	1.0	1.25	1.5	2.0	2.5

Table 2: Inflow List

The duration of each simulation was scheduled to last 1 hour in simulation time. Thus, steady-state observations are assured and reliable conclusions can be drawn. Furthermore, measurements were taken every 5 seconds and signal heads phases were switched every 20 seconds. These values of sampling time and phase cycle allow having a good balance between computational performance and results accuracy. The simulation resolution was set at 1 second.

The network (Figure 4) is comprised of more than 1000 links. Each link has a different length, and the number of lanes can vary from 1 to 4. Furthermore, the number of links analyzed is also related to the computational time needed during the simulations: the lower the number of links checked, the higher the time needed to run a single simulation. For this reason, only lanes used by the Max-Weight algorithm were checked during the simulation. A converted MATLAB table (.mat) was employed to import this data. See appendix B for further information.

3.2.2 Simulation methodology

In order to run the model, PTV Vissim was executed using the MATLAB environment. The communication between software was achieved thanks to Vissim's COM interface (Component Object Model interface). COM interface enables inter-process communication object creation in a large range of programming languages [16][17]. In this case, COM was used to launch the simulation, modify parameters, and access the data.

In this context, two types of main MATLAB scripts (programs) were considered: the ones that run the simulation and save the data, one per case (four in total), and the ones that plot the results.

At the beginning of each iteration (step), during the simulation, several variables are checked per lane: queue, speed, the volume of cars, etc. Yet, as explained above, only lanes required in the Max-Weight algorithm were used so as to speed up calculations. Once all the necessary data is collected, the MW algorithm is run, and the most suited signal head phase per controlled junction is selected.

For cases three and four, noise is added to the queue measurements. In addition, the fourth case implements the particle filter and it is applied essentially as in [8]. The main changes can be found in the prediction part where, in order to avoid the effect of the accumulated error as the simulation advances, the particles were obtained thanks to a truncated normal distribution. See appendix C for further information about the code.

3.3 Data collection and plotting

As explained above referring to data acquisition, the majority of the data was taken from the lanes used by the MW algorithm, where information about the vehicles could be extracted: speed, delay, time travel, and therefore the volume of vehicles and queue. Local mean values of flow, density and speed could be obtained thanks to that method; however, data collections points, which they were also implemented, were more suitable for this application.

Queue detectors were also placed next to the controlled junctions, but it was determined, that using the data given by the lanes, was more computationally efficient.

Overall NFD data was obtained thanks to general network parameters, as for example, the number of departed vehicles (from the starting point) per simulation step, or the number of vehicles that left the network. By knowing both variables, the actual number of vehicles in the network could be deduced.

At the end of the simulation, desired variables and parameters were saved in a *.mat* MATLAB file. In this way, simulation data could be checked freely at any time.

Finally, graphs were obtained separately by different MATLAB scripts. All the results are analyzed in the following section. See appendix C for further information about the scripts.

4 Results and discussion

4.1 NFD

In order to compare network capacities between fixed signal schedules and Max-Weight algorithm control, the Network Fundamental Diagram was studied for both cases. Figure 5 displays the NFD for case 100, where fixed and cyclic signal heads control was used, whereas figure 6 shows the NFD for case 425 with MW Control, noise and particle filter. Both cases used the same OD matrix and flow represents the vehicles exiting the network.

As it can be seen, both figures represent the shape discussed in section 2: the free-flowing part which is the line with positive slope, and the congested part, which has a negative slope and where points are more scattered.

Both cases have approximately the same positive slope. Regarding the maximum point of traffic flow ρ_c, which in both cases is 140 vehicles/min, the maximum is attained at 500 vehicles for the case 100 , whereas for the case 425 it is reached at 700 vehicles. This means that the network using the Max-Weight algorithm control is capable of handling and absorbing a major quantity of vehicles than the fixed signal schedules network.

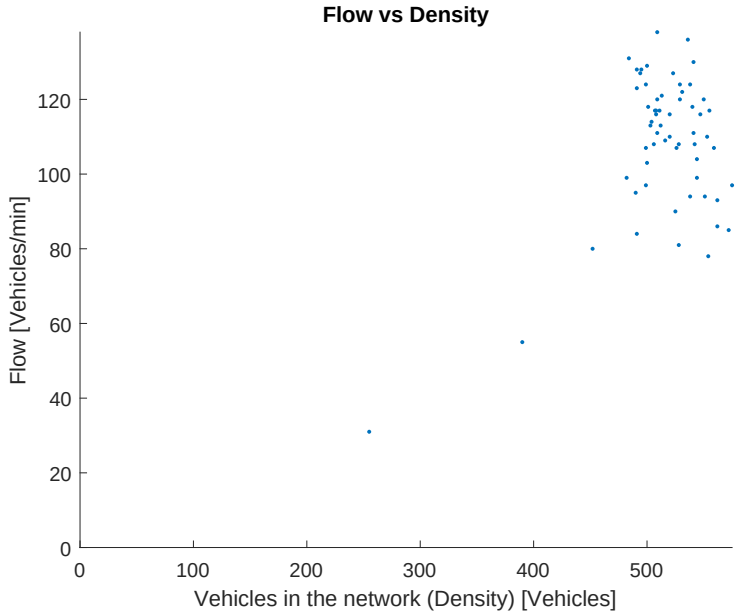

Figure 5: General NFD, case 100 (Fixed and cyclic signal heads), inflow=2.0 .

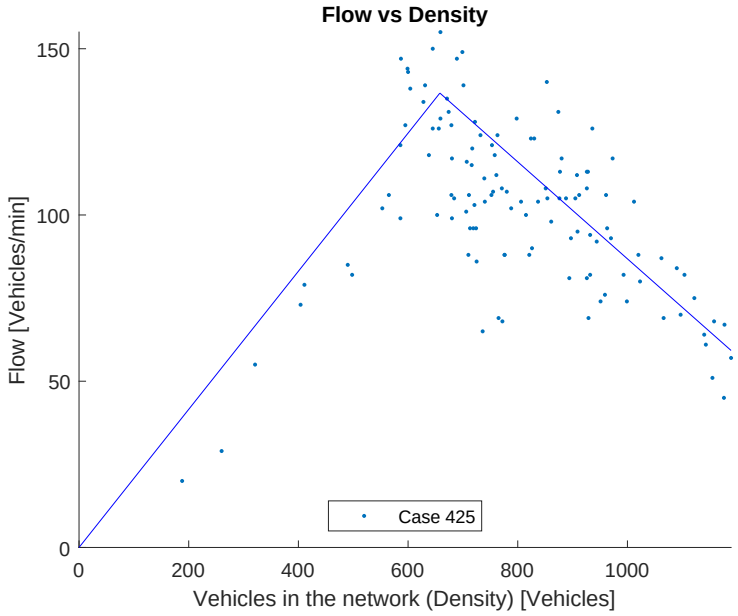

Figure 6: General NFD, case 425 (Max-Weight Control, noise=25%, filtered), inflow=1.5, 2.0, 2.5.

4.2 Overall results

After analyzing the NFD graphs, overall results were studied. Figures 7 and 7 display all the data collected about the queue, speed, travel time and departed vehicles, depending on vehicle inflow. In every chart, four graphs are represented, one per case. The difference between both figures is the amount of noise added in queue measurements for cases three and four, 10% or 25%. It is important to note that having the same inflow value between cases does not necessarily mean that they had used the same OD matrix during the simulation: only the amount of cars that entered the network in one hour (whole simulation) were counted as inflow vehicles.

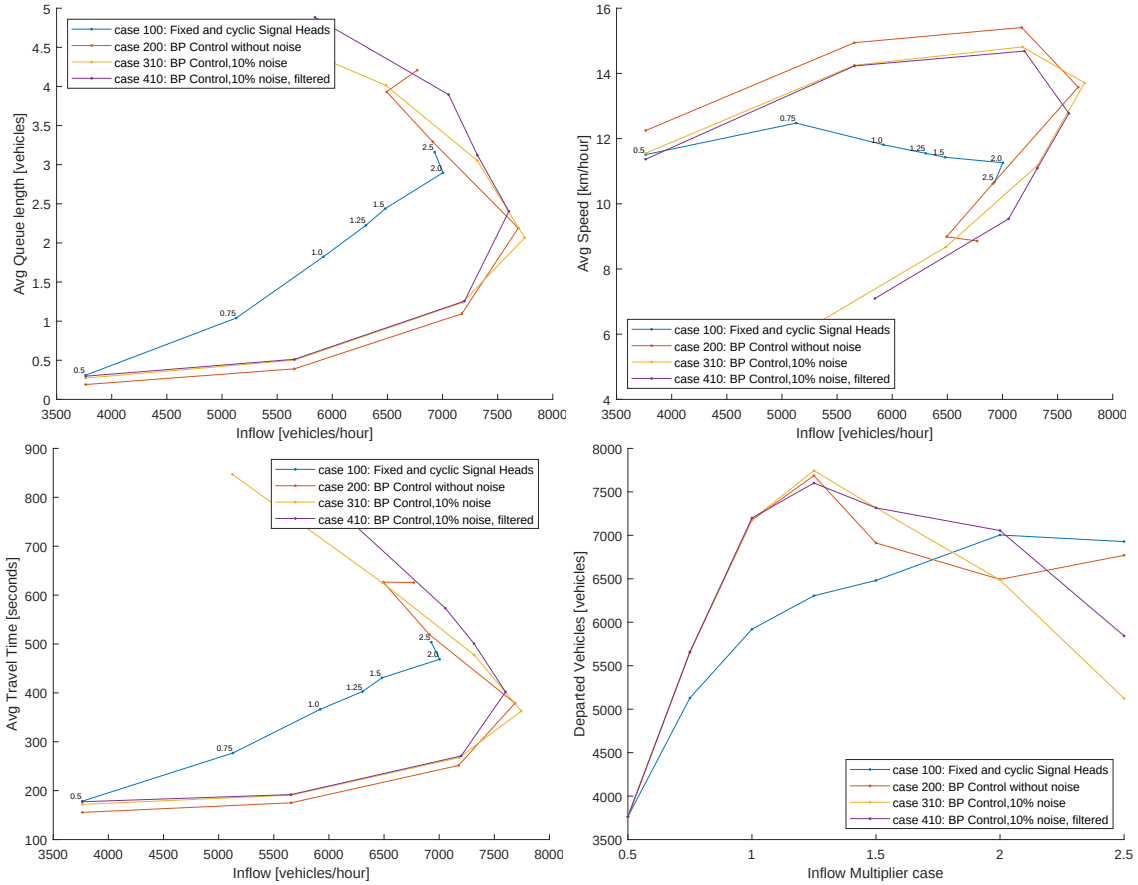

Figure 7: Overall Results. Cases: 100, 200 (MW alg.), 310 (MW alg., noise=10%) and 410 (MW alg., noise=10%, filtered). Numbers next to the points correspond to the OD matrix multiplier. Top-left: average queue length per vehicle inflow. Top-right: average speed per vehicle inflow. Bottom-left: average travel time per vehicle inflow. Bottom-right: total departed vehicles per vehicle inflow.

Considering figure 7, where cases 100, 200, 310 and 410 (noise = 10%) are represented, a characteristic feature can be noticed in the four charts: the inflow maximum capacity. As noted above, the maximum inflow handled by the Max-Weight algorithm (cases 200, 310, 410), approximately 7500 vehicles per hour, is higher than the fixed signal schedules implementation (case 100), which is around 6800 vehicles

per hour.

Furthermore, the performance of the Max-Weight algorithm is proven: according to figures 7 and 8, fixed signal heads control is significantly superior to the Max-Weight algorithm control in terms of both average queue length and travel time at vehicles rates under the critical maximum inflow. They also show that the average speed of vehicles and the number of vehicles departed from their origin are mainly highest under the MW algorithm.

As can be seen in queue and travel time charts, graphs follow a "U" shape. This is due to the fact that when MW algorithms are employed over a network with finite length links, some roads can experience a deadlock situation. In other words, when the inflow rate is greater than the critical maximum inflow, at very high traffic loads, bottlenecks and deadlocks are formed until the network get saturated. At the end of the simulation, because of the bottlenecks, the total amount of vehicles served is lower than the maximum capacity.

By focusing on the Max-Weight algorithm controllers, the case 200, which worked in ideal conditions, without noise in the measurements, it's the one that clearly had the best performance. If cases 310 and 410 are compared, no evident difference can be detected. Nevertheless, the case with a particle filter (410) seems to be more sensitive to deadlocks at very high traffic loads.

Finally, if noise is increased up to 25%, as it can be appreciated in figure 8, case 425 performs slightly better than 325 at medium-high traffic loads.

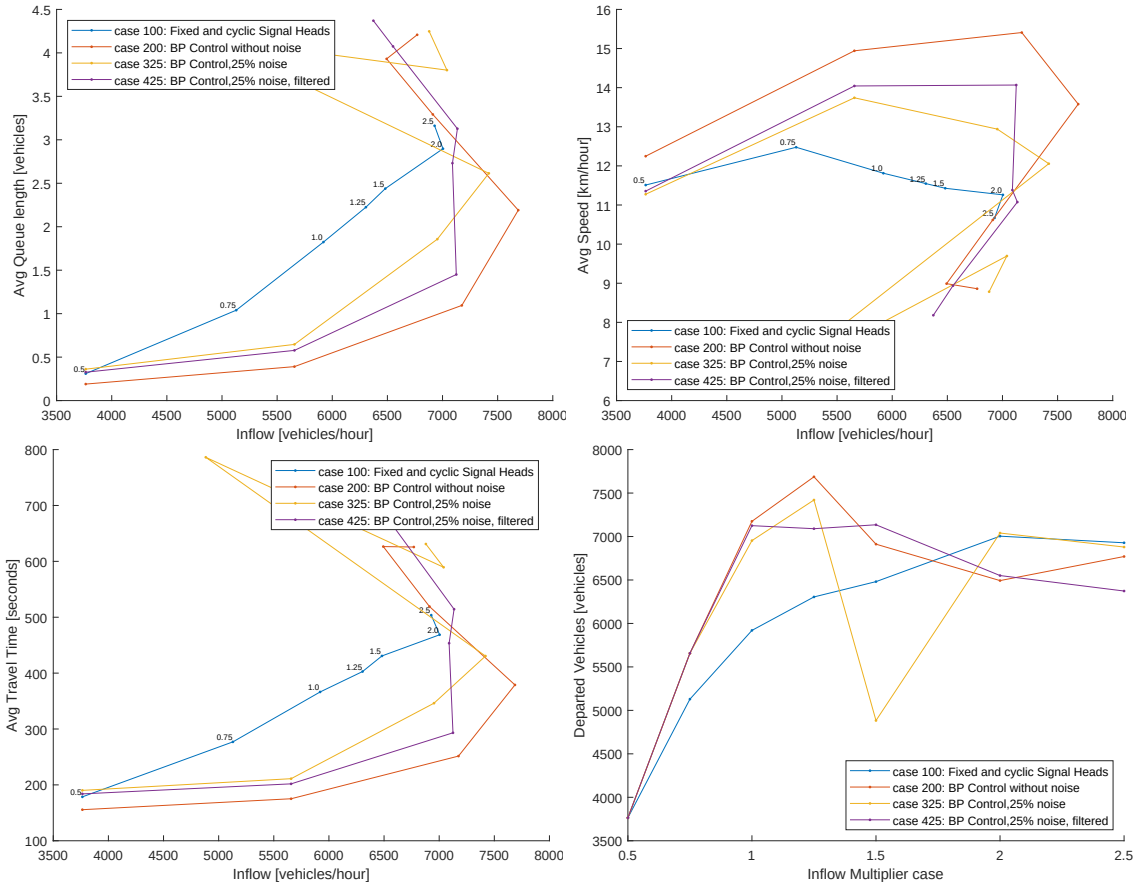

Figure 8: Overall Results. Cases: 100, 200 (MW alg.), 325 (MW alg., noise=25%) and 425 (MW alg., noise=25%, filtered). Numbers next to the points correspond to the OD matrix multiplier. Top-left: average queue length per vehicle inflow. Top-right: average speed per vehicle inflow. Bottom-left: average travel time per vehicle inflow. Bottom-right: total departed vehicles per vehicle inflow.

4.3 Case: Rush hour inflow - same OD matrix

In this section, only the rush hour inflow (inflow = 1.0) was considered. All the simulations were obtained with the same OD matrix. This means that the real rate of vehicles entering the network could be different between cases, as demonstrated above.

Figure 9 displays the sum of queues per the simulation time, and it is very useful to study the evolution of the network during the simulation. Thanks to this figure and figure 10, which shows the mean and the standard deviation of the queue length, it can be deduced that the Max-Weight algorithm without noise (case 200) in the measurements has the best performance, even with higher inflow rate than fixed signal scheduled control.

No difference is appreciated between cases 310 and 410. However, in figures 11 and 12, where noise is increased, the MW control with noisy measurements filtered by a particle filter (case 425), obtained better results than the unfiltered case (325).

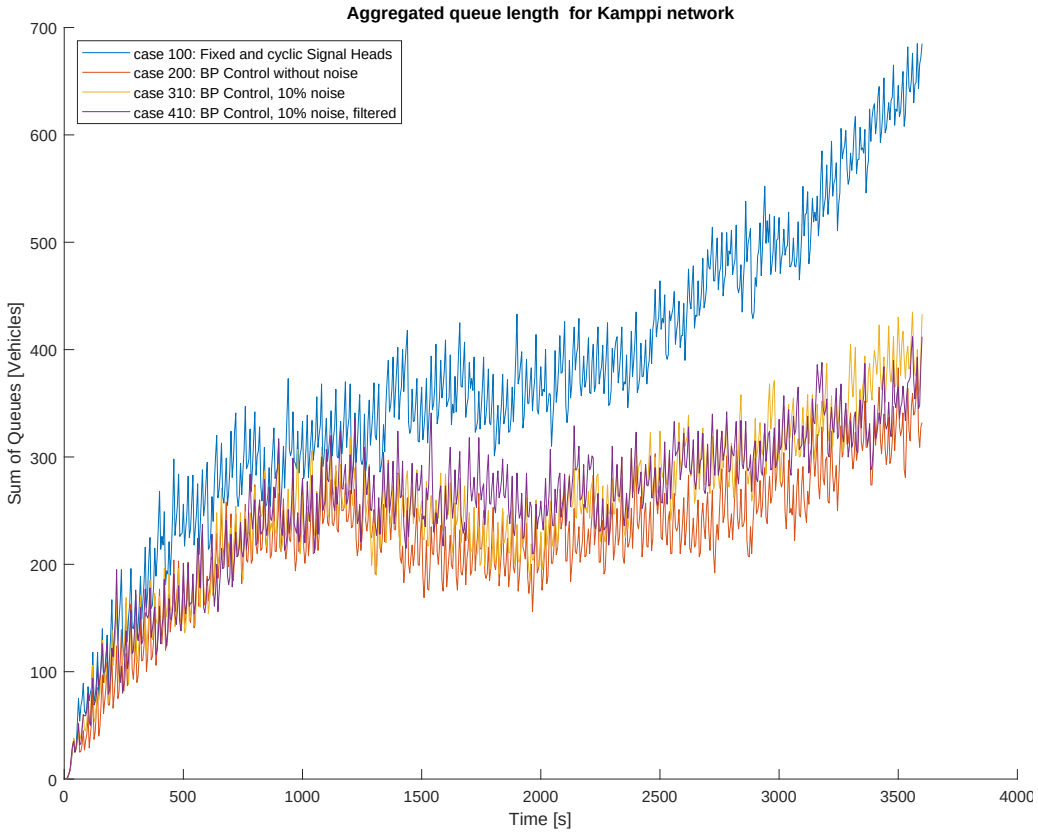

Figure 9: Sum of queues per simulation time. Cases: 100, 200 (MW alg.), 310 (MW alg., noise=10%) and 410 (MW alg., noise=10%, filtered).

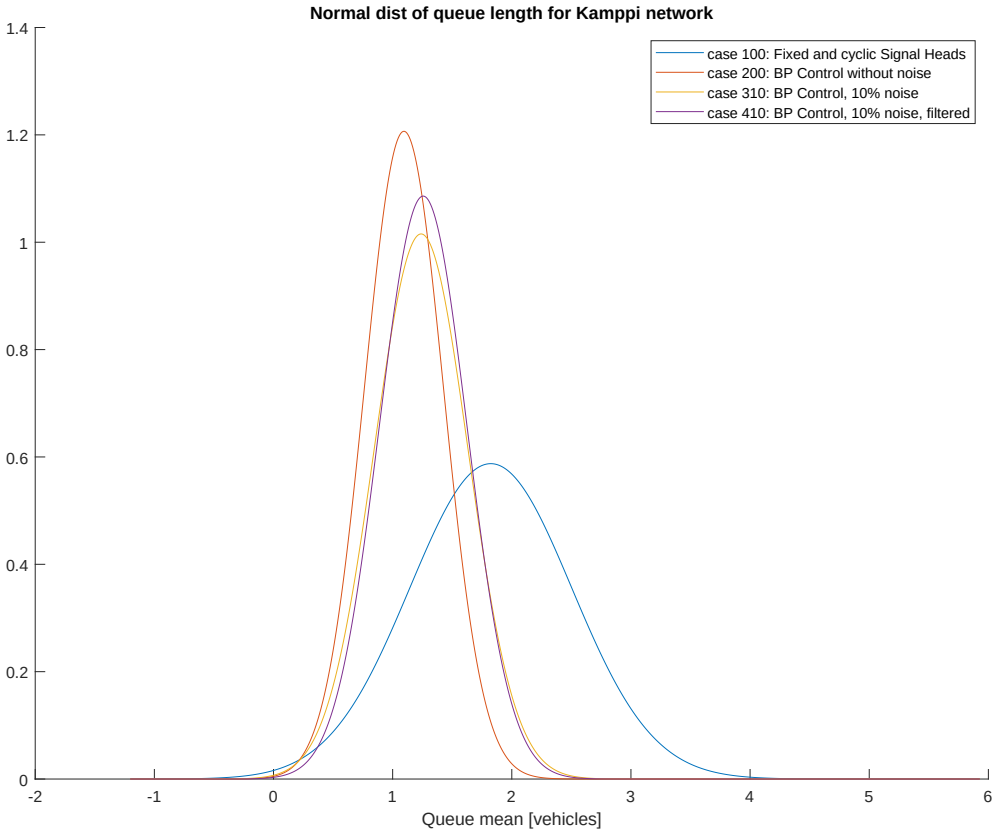

Figure 10: Normal distribution of the queue length. Cases: 100, 200 (MW alg.), 310 (MW alg., noise=10%) and 410 (MW alg., noise=10%, filtered).

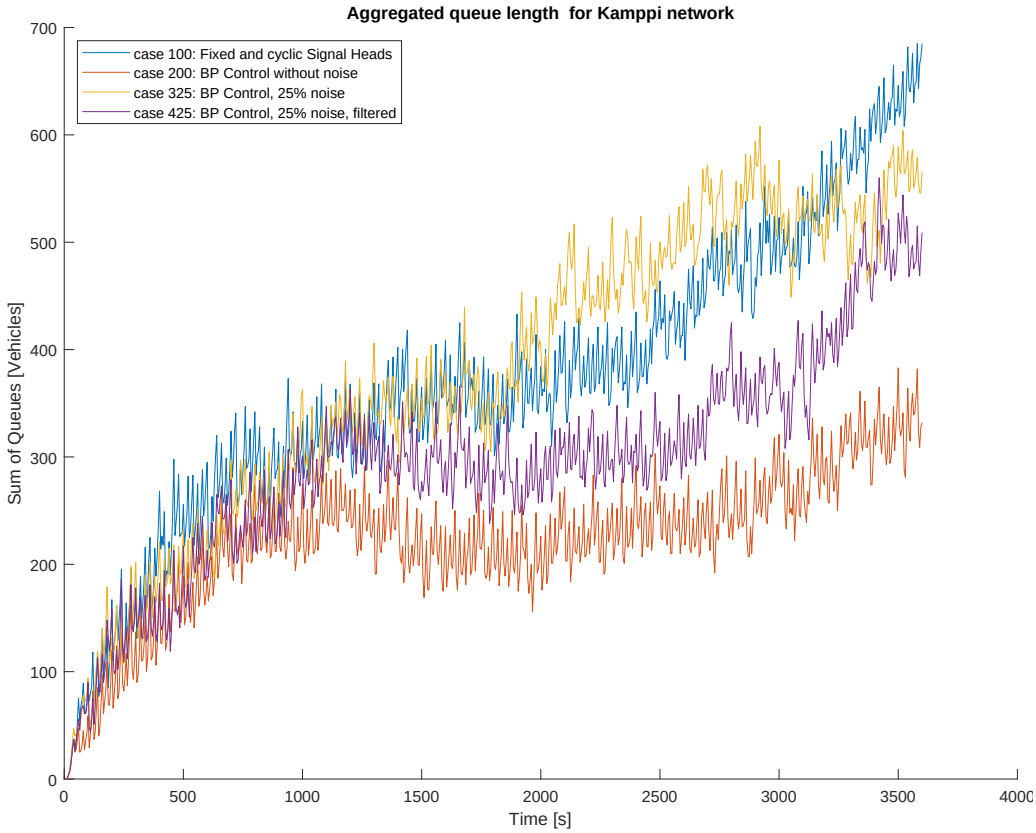

Figure 11: Sum of queues per simulation time. Cases: 100, 200 (MW alg.), 325 (MW alg., noise=25%) and 425 (MW alg., noise=25%, filtered).

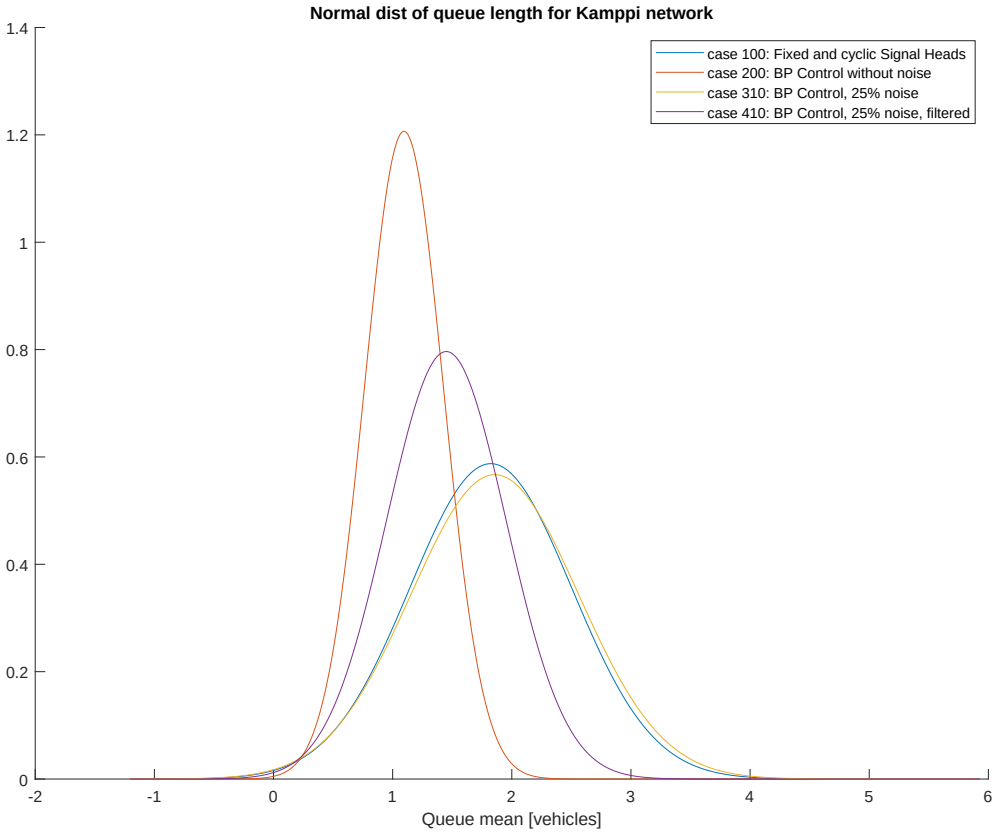

Figure 12: Normal distribution of the queue length. Cases: 100, 200 (MW alg.), 325 (MW alg., noise=25%) and 425 (MW alg., noise=25%, filtered).

4.4 Case: Rush hour inflow - 7000 veh/hour

In contrast to the last section, the rush hour inflow rate, 7000 vehicles per hour, was taken into account, rather than using the same OD matrix for all the cases. Figure 13 shows the chosen cases for this analysis, in green.

Thanks to figures 14 and 15, the performance of the Max-Weight algorithm is demonstrated. By having the same rate of vehicles entering the network, the fixed and cyclic control clearly obtained greater queues values.

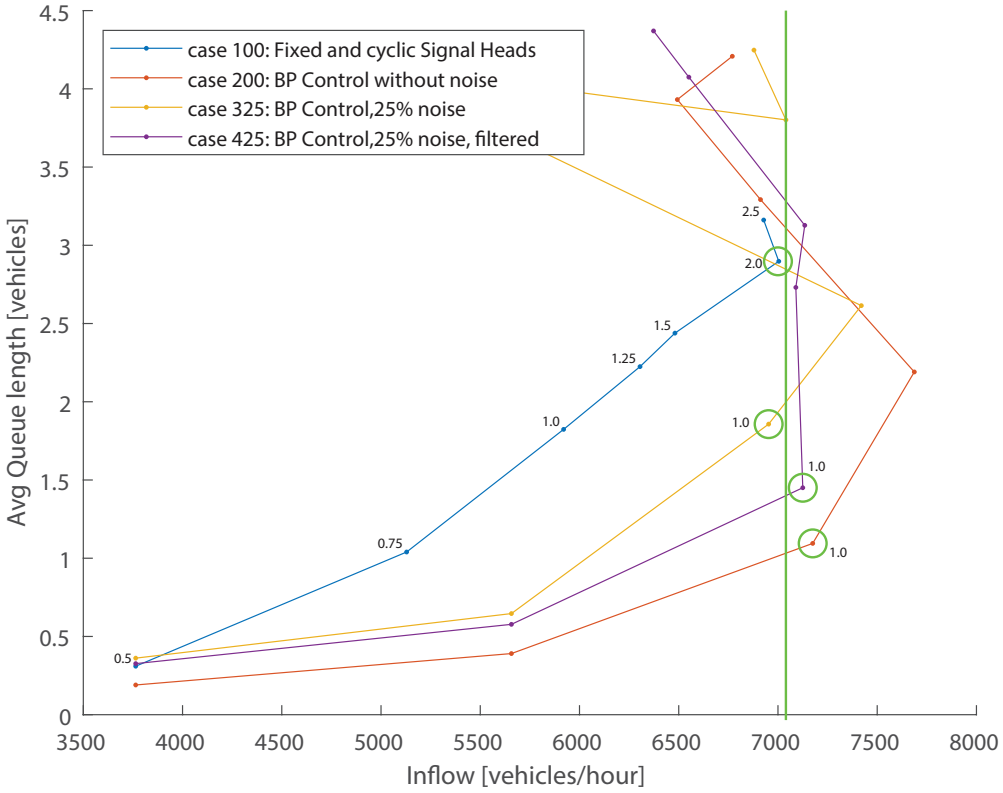

Figure 13: Average queue length per vehicle inflow. Cases: 100, 200 (MW alg.), 325 (MW alg., noise=25%) and 425 (MW alg., noise=25%, filtered). Numbers next to the points correspond to the OD matrix multiplier. In green, the cases chosen to represent aprox. 7000 veh/hour inflow.

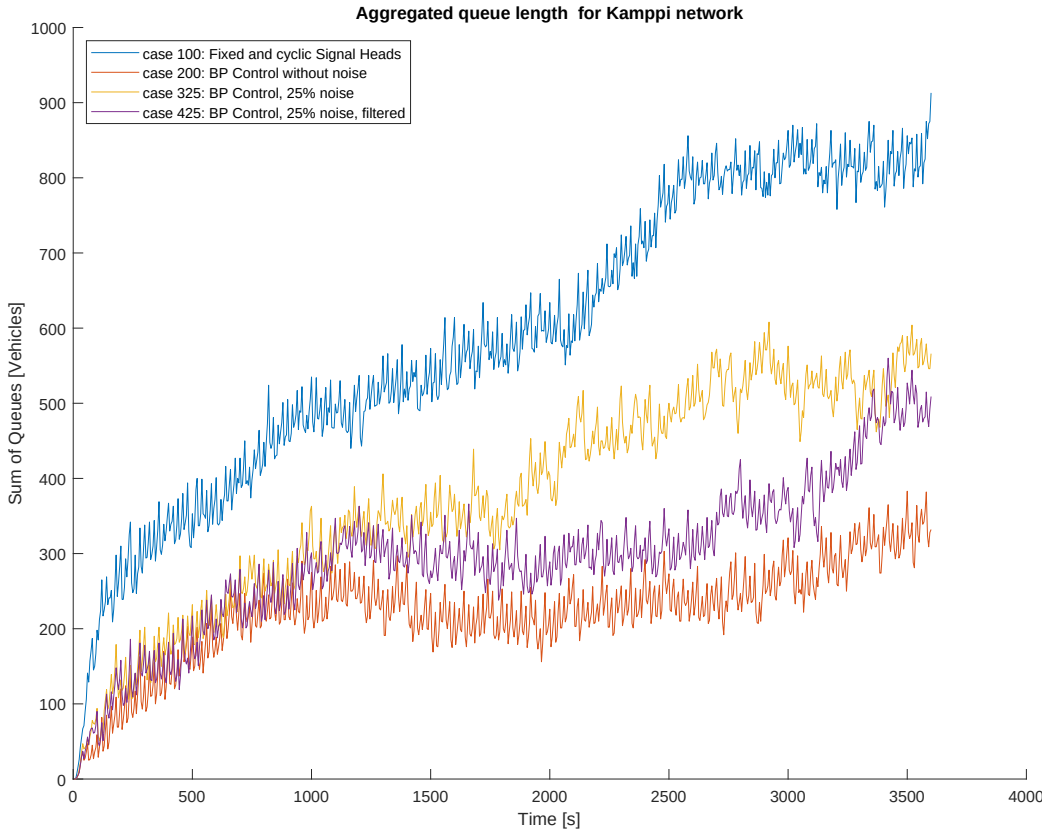

Figure 14: Sum of queues per simulation time. Cases: 100, 200 (MW alg.), 325 (MW alg., noise=25%) and 425 (MW alg., noise=25%, filtered).

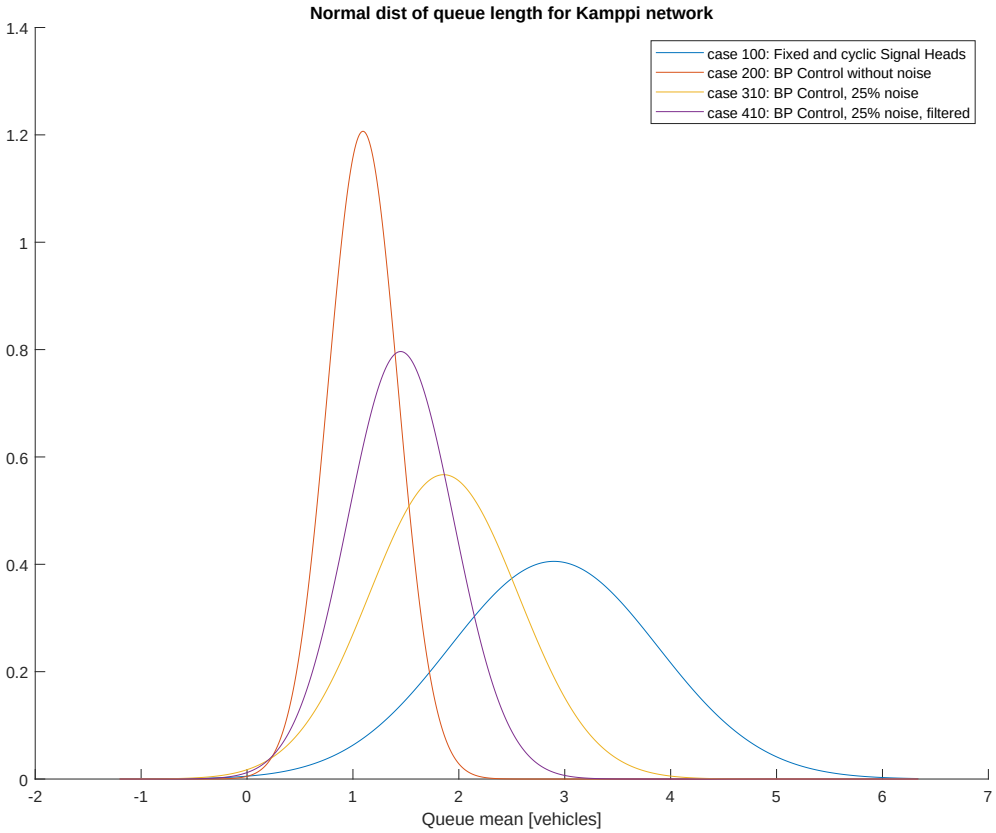

Figure 15: Normal distribution of the queue length. Cases: 100, 200 (MW alg.), 325 (MW alg., noise=25%) and 425 (MW alg., noise=25%, filtered).

5 Conclusions and future directions

This thesis examined a part of Helsinki's city center network to verify theories of traffic flow and tested the performance of the Max-Weight algorithm under different scenarios.

The main conclusion that can be drawn is that the Max-Weight algorithm control outperforms the traditional fixed and cyclic control under rush hour situation in Helsinki. Under the same traffic conditions (same demand), besides improving queue, average speed, and travel time conditions, the Max-Weight algorithm control is capable of handling higher traffic loads without collapsing the network.

Nevertheless, under noisy measurements, the tested filtered Max-Weight algorithm control did not make much difference, compared with the non-filtered noisy control. This can be partly explained by the fact that the implemented particle filter is an adaptation of the one proposed in [8] and did not strictly follow all the requirements.

In future investigations, more research is needed to apply and test the filtered Max-Weight algorithm control in the PTV Vissim model, and achieve the expected results. Further studies should investigate the performance of the algorithm in larger networks, such as all Helsinki center, not only Kamppi neighborhood. In addition, phase duration and phase activation parameters might prove an important area for future research, in order to study how they affect the performance of the algorithm.

Finally, future research should examine different types of networks: more dense and organized networks, as the Eixample Neighborhood in Barcelona, or on the contrary, a less homogenous network, as for example, an old city center, and how the reliability of the algorithm is affected by the geometry of the network.

References

[1] PTV Group, *PTV Vissim*. Available: https://www.ptvgroup.com/en/solutions/products/ptv-vissim/ [Accessed: June, 2019].

[2] R. Savolainen, *Modeling and analyzing Helsinki's traffic network using a microscopic simulator*, B.S. thesis, Aalto University, Espoo, FI, May 2018.

[3] J. Popping, *An overview of microscopic and macroscopic traffic models*, Groningen, NL, B.S. thesis, July 2013. Available: http://fse.studenttheses.ub.rug.nl/11050/1/Bachelorproject.pdf [Accessed: June, 2019].

[4] M. J. Park, *Three phase traffic theory*, December 2012. Available: http://guava.physics.uiuc.edu/~nigel/courses/569/Essays_Fall2012/Files/Park.pdf/ [Accessed: June, 2019].

[5] B. S. Kerner, *The Physics of Traffic: empirical freeway pattern features, engineering applications, and theory*. Berlin: Springer cop., 2004.

[6] Y. Sugiyama, M. Fukui, M. Kikuchi, K. Hasebe, A. Nakayama, K. Nishinari, S. Tadaki and S. Yukawa, "Traffic jams without bottlenecks—experimental evidence for the physical mechanism of the formation of a jam", *New Journal of Physics*, vol. 10, March 2008. Available: https://iopscience.iop.org/article/10.1088/1367-2630/10/3/033001/meta [Accessed: June, 2019].

[7] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Transactions on Automatic Control*, vol. 37, no. 12, pp. 1936–1948, Dec. 1992.

[8] T. Charalambous, H. Farhadi, M. Taki, B. Kulcsár, and H. Wymeersch, "Back-pressure traffic signal control with noisy queue information".

[9] M. J. Neely. "Notes on Backpressure Routing",*EE 649 Stochastic Network Optimization*, Spring 2011. Available: https://www.win.tue.nl/~sem/2MMS40/Chapter8.pdf [Accessed: June, 2019].

[10] T. Bonald, A. Proutière (2003). Insensitive bandwidth sharing in data networks Queueing Systems 44 (1), 69–100. Available: https://www.win.tue.nl/~sem/2MMS40/Chapter8.pdf [Accessed: June, 2019].

[11] A. Zaidi, B. Kulcsar, H. Wymeersch (2016), "Back Pressure Traffic Signal Control with Fixed and Adaptive Routing for Urban Vehicular Network". *IEEE transactions on intelligent transportation systems*, vol. 17(8), pp. 2134-2143. Available: http://publications.lib.chalmers.se/records/fulltext/231155/local_231155.pdf [Accessed: June, 2019].

[12] OpenStreetMap contributors, "Planet dump retrieved from https://planet.osm.org". Available: https://www.openstreetmap.org [Accessed: June, 2019].

[13] PTV Group, *PTV Vissim 11 User Manual*. PTV AG, Karlsruhe, Germany, 2018.

[14] U. E. D. City of Helsinki, "Liikennemäärät helsingissä". Available: https://www.avoindata.fi/data/fi/dataset/liikennemaarat-helsingissa [Accessed: April, 2019].

[15] Google Maps, "Helsinki traffic state". [Accessed: April, 2019].

[16] Microsoft, "What Is a COM Interface?". Available: https://docs.microsoft.com/en-us/windows/win32/learnwin32/what-is-a-com-interface- [Accessed: May, 2019].

[17] T. Tettamanti, "A practical manual for Vissim COM programming in Matlab - 3rd edition for Vissim version 9 and 10", January 2018. Available: https://www.researchgate.net/publication/322232068_A_practical_manual_for_Vissim_COM_programming_in_Matlab_-_3rd_edition_for_Vissim_version_9_and_10 [Accessed: May, 2019].

[18] Enciclopèdia.cat, "L'Eixample de Barcelona" https://www.enciclopedia.cat/EC-GEC-0023666.xml [Accessed: May, 2019].

A OD Matrix

Table A1 shows the Origin-Destination Matrix used during the simulations. First column: origins. First row: destination. The matrix represents the number of vehicles that travel from an origin to a destination during 30 minutes of the simulation.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	Total
1	0	13	13	13	45	21	120	115	70	25	13	0	13	13	0	13	64	0	548
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	20	2	0	2	8	2	41	10	20	10	10	0	10	10	0	2	15	0	162
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	41	6	6	6	0	6	58	16	41	25	15	0	15	15	0	6	33	0	293
6	16	2	2	2	8	0	20	20	13	5	2	0	2	2	0	2	12	0	112
7	132	19	19	19	66	19	0	16	16	38	16	0	25	19	0	19	104	0	527
8	132	19	19	19	66	19	16	0	104	38	19	0	19	19	0	19	104	0	610
9	68	11	11	11	37	11	16	95	0	22	11	0	11	11	0	11	53	0	378
10	41	4	4	4	15	4	40	40	24	0	4	0	4	4	0	4	25	0	220
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	14	2	2	2	8	2	19	19	12	4	2	0	2	2	0	2	11	0	105
13	16	2	2	2	8	2	20	20	13	5	2	0	0	2	0	2	12	0	112
14	23	4	4	4	12	4	31	31	19	7	4	0	4	0	0	4	18	0	167
15	23	4	4	4	12	4	31	31	19	7	4	0	4	4	0	4	18	0	170
16	14	2	2	2	8	2	19	19	12	4	2	0	2	2	0	0	11	0	102
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	62	10	10	10	37	16	91	86	52	19	10	0	10	10	0	10	48	0	480
Total	602	100	98	100	330	112	522	519	417	209	114	0	121	114	0	98	528	0	3984

Table A1: Oringin-Destination Matrix

B Network Data Table

The main function of this table is to specify the links and lanes used in the Max-Weight algorithm. Every row represents a signal controller (a controlled junction). For each controller, the number of phases has to be noted, as well as the complexity of the junction: if a link is composed of more than one lane that goes to the same lane, it is considered complex. If a long street has to be controlled and it is composed of several links, in order to check the complete queue, both links have to be counted, and it is considered complex. See the table B2 for more information. Furthermore, for each controller, the links and lanes used for each phase have to be stated.

Example:

Signal Controller	Phases	Complex	Ph 1 IN Links	Ph 1 IN Lanes	Ph 1 OUT Links	Ph 1 OUT Lanes
24	2	1	[137 137 244 244;137 0 244 244]	[1 2 1 2;3 0 1 2]	[41 41;376 376]	[1 2;1 2]

Table B1: Example - Controller 24

In this example, only the first phase is considered. Each phase is composed of four columns: two for the lanes entering the junction and two for the lanes exiting the junction. "Ph 1 IN Links" column and "Ph 1 IN Lanes" are paired, as well as the OUT columns. Each cell is a MATLAB matrix. Both matrices of IN column cells have the same dimension: each position in the matrix corresponds to a concrete link and lane. If we consider the first position in the "Ph 1 IN Links" and "Ph 1 IN Lanes" matrices, it means that one of the lanes observed is the first lane of the link number 137. The next one, it is the second lane of the link 137. Third column, first row of the matrices: lane 1 of the link 244. If a phase is composed of multiple stages (different income and outgoing possibilities that can be served in the same phase), each row corresponds to a different stage, considering the matrix. Between IN and OUT columns, matrices need to have the same number of rows (stages), but their dimensions can be different.

Channel	Controller	Phases	Cycle(sec)	Phi_1(N) Matrix	Phi_2 (N) Matrix	Phi_3 (N) Matrix	Phi_4 (N) Matrix	Phi_5 (N) Matrix	Phi_6 (N) Matrix	Phi_7 (N) Matrix	Phi_8 (N) Matrix	Phi_9 (N) Matrix	Phi_10 (N) Matrix	Phi_11 (N) Matrix	Phi_12 (N) Matrix	Phi_13 (N) Matrix	Phi_14 (N) Matrix	Phi_15 (N) Matrix	Phi_16 (N) Matrix
1	1	2	0	[1,1]	[1,2]	[1,2,4]	[1,2,4]	[1,1,2,4]											
2		2	0	[1,2,4]	[1,1]	[1,2,4]	[1,2,4]	[1]											
3		2	0	[1,1,1,1]	[1,1,1,1]	[1,1,1,1]	[1,1,1,1]	[1,1,1,1]											
4	2	2	0	[17,17,1,17,17]	[1,1,1]	[99,99,272,272]	[99,294,294,294]	[1,1,1,1]											
5	3	2	0	[11,11,1,11,11]	[1,1,1,1]	[99,99,272,272]	[99,294,294,294]	[1,1,1,1]											
6	3	2	0	[10,0,0,0]	[1,1,1,1,1,4]	[33,33,77,77,117,304]	[233,272,272,272]	[1,1]											
7	4	2	0	[17,17,1,17,17,137,137]	[1,1]	[33,33,77,77,117,304]	[233,238,238,238]	[1,1]											
8	2	2	0	[69,69,275,275]	[1,1,1,1]	[99,99,272,272]	[99,294,294,294]	[1,1,1,1]											
9	3	2	0	[69,69,275,275]	[1,1,1,1]	[99,99,272,272]	[99,294,294,294]	[1,1,1,1,4]											
10	2	2	0	[22,22,2,22,22]	[1,1,2]	[17,69,69]	[17,69,69]	[1,1,1,4]											
11	2	2	0	[22,22,2,22,22,137]	[1,1,2,4]	[17,69,69]	[17,69,69]	[1,1,1,4]											
12	2	2	0	[99,99,99,99]	[1,2]	[227,227,272]	[263,263,263]	[1,1,2]											
13	2	2	0	[99,99,99,99]	[1,1,2,4]	[227,227,272,304]	[263,263,263]	[1,1,2]											
14	2	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
15	2	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
16	2	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
17	2	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
18	2	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
19	2	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
20	2	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
21	2	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
22	2	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
23	2	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
24	2	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
25	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
26	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
27	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
28	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
29	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
30	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
31	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
32	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
33	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
34	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
35	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
36	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
37	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
38	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
39	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
40	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
41	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
42	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
43	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
44	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
45	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
46	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
47	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
48	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
49	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
50	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
51	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
52	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
53	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
54	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
55	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
56	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
57	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
58	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
59	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
60	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
61	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
62	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
63	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
64	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
65	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
66	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
67	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
68	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
69	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
70	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
71	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
72	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
73	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
74	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
75	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
76	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
77	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
78	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
79	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
80	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
81	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
82	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
83	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
84	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
85	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
86	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
87	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
88	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
89	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
90	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
91	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
92	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
93	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
94	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
95	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
96	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
97	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
98	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
99	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
100	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
101	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
102	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
103	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
104	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
105	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
106	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
107	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
108	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
109	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
110	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
111	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
112	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
113	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
114	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
115	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
116	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
117	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
118	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
119	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
120	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
121	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
122	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
123	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
124	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
125	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
126	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
127	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											
128	3	2	0	[99,99,99,99]	[1,1,2,4]	[33,33,77,77]	[33,33,33,33]	[1,2,1,1]											

Table B2: Network Data table

C MATLAB code

C.1 Main Simulation Script

C.1.1 Noise and particle filter implementation

For cases three and four, noise is added to the queue measurements as it follows:

```
% NL = Noise Level
NL(i,k)=round(Link_length(i)/5*noise);
% Average car length = 5m

% QN = Noisy Queue
QN(i,k)=max(Q(i,k)+randi([-NL(i,k) NL(i,k)],1,1),0);
```

$NL(i,k)$ is the noise level, or in other words, the maximum absolute error that a lane can attain per lane i and second k. The variable *noise* corresponds to the noise factor (in this case, 0.1 or 0.25). As may be seen, $NL(i,k)$ is the average maximum capacity of vehicles that the lane i can achieve, multiplied by the noise factor. The noisy queue $QN(i,k)$ is obtained by the uniformly distributed pseudorandom integers function *randi*, using $NL(i,k)$ as boundary.

For the fourth case, the particle filter was applied essentially as in [8]. The main changes can be found in the prediction part where, in order to avoid the effect of the accumulated error as the simulation advances, the particles were obtained thanks to a truncated normal distribution, as it follows:

```
% Prediction
pd = makedist('Normal','mu',0,'sigma',4);
tpd = truncate(pd,-5,5);
tpdm = round(random(tpd,1,100));
q_pred(i,:)=Q(i,k)+tpdm;
```

C.1.2 The script

```
% This program implements Back-Pressure signaling control algorithm with
% noisy queue state information for Helsinki network
% CASES:
% case 1: Fixed and cyclic Signal Heads - without control (Nominal case)
% case 2: Back-pressure traffic signal control without noise
% case 3: Back-pressure traffic signal control with noise
% case 4: Back-pressure traffic signal control with noise and filtered

clear all;
clc;

num_case=4; % Case Number

inflow_values=[0.5,0.75,1.0,1.25,1.5,2.0,2.5]; % Inflow value multiplier

particles=100; % Number of particles used by the filter
noise_values=[0.1,0.25]; % Noise level
% Create/Verify folders
```

```
if exist('results_sims','dir')==0
    mkdir('results_sims');
end
if exist('results_sims\screenshots','dir')==0
    mkdir('results_sims\screenshots');
end

%%
for noise=noise_values
    for inflow=inflow_values

        tic
        close all;

        % Show Message 1 & 2
        message=strcat('Case: ',num2str(num_case),' | Noise: ',num2str(noise*100),'
            %',' | Inflow: ',num2str(inflow),' | ',datestr(datetime));
        disp(message)
        disp('Initializing...')

        %---------------------------------------------------------------
        % IMPORT DATA

        load('kamppi_network_sc.mat');          % Load SC Network Properties
        load('kamppi_lanes_table_sc.mat');      % Load SC Network Properties
        load('arrivals.mat');                   % Load SC Network Properties

        %---------------------------------------------------------------
        % SIMULATION DATA


        Tsig=20;                        % Control Period Time
        seconds=3600;                   % 3600 (simulation time in seconds)
        sampling_time=5;

        %---------------------------------------------------------------
        % INITIALIZE
        vissim_com=actxserver('Vissim.Vissim.1100');
        vissim_com.LoadNet([pwd '\kamppi_sc_1.inpx']);

        sim=vissim_com.Simulation;
        vnet=vissim_com.Net;
        vehs=vnet.Vehicles;
        scs=vnet.SignalControllers;
        links=vnet.Links;
        demand_matrix1=vnet.DynamicAssignment.DynAssignDemands.ItemByKey(1).Matrix;
        demand_matrix2=vnet.DynamicAssignment.DynAssignDemands.ItemByKey(2).Matrix;

        LinkID_list=lanes_table_sc{:,1}';
        LaneID_list=lanes_table_sc{:,2}';

        max_phases=max(network_sc.Phases);
        ind=ones(1,size(network_sc,1));
        prev_ind=ones(1,size(network_sc,1));
```

```
72
73          Q=zeros(size(lanes_table_sc,1),seconds); % Queue
74          QN=zeros(size(lanes_table_sc,1),seconds); % Queue with noise
75          QNF=zeros(size(lanes_table_sc,1),seconds); % Queue with noise filtered
76          Volume_t=zeros(size(lanes_table_sc,1),seconds); % Traffic Volume
77          Speed=zeros(size(lanes_table_sc,1),seconds); % Vehicle Speed (SUM)
78          Avg_Speed=zeros(size(lanes_table_sc,1),seconds); % Vehicle Speed (AVG)
79          NL=zeros(size(lanes_table_sc,1),seconds); % Noise
80          prediction=zeros(size(lanes_table_sc,1),seconds); % Prediction (Filter)
81
82          delaytime=zeros(size(lanes_table_sc,1),seconds); % Sum of delay time (per
                sampling time)
83          timeinnetwork=zeros(size(lanes_table_sc,1),seconds); % Sum of time in
                network (per sampling time)
84          vehdeparted=zeros(1,seconds); % Sum of departed vehs in network (per
                seconds or step)
85          veharrived=zeros(1,seconds); % Sum of arrived vehs in network (per seconds
                or step)
86          vehevolution=zeros(1,seconds); % Number of vehs that are in the network or
                planned to enter the network
87
88          vehicles_data_temp=cell(size(lanes_table_sc,1),seconds); % Record of delay
                and time per per lane and vehicle
89
90          % Converts INPUT: link and lane to OUTPUT: position in matrix
91          % (for example, it returns the Q matrix index for a concrete
92          % lane and link)
93          % Be careful here with the maximun number of lanes in the network, here 4
94          conversion_list=zeros(size(lanes_table_sc,1),4);
95
96          for i=1:size(lanes_table_sc,1)
97              conversion_list(LinkID_list(i),LaneID_list(i))=i;
98          end
99
100         % Signal Groups (per Signal Controller)
101         sgs={};
102         for j=1:size(network_sc,1)
103             sgs{j}=scs.ItemByKey(j).SGs;
104         end
105
106         % Record of State of Signal Heads
107         Signal_heads_states=ones(size(network_sc,1),max_phases,seconds);
108         for k=1:seconds
109             for j=1:size(network_sc,1)
110                 if network_sc{j,2}~=max_phases
111                     for i=network_sc{j,2}+1:max_phases
112                         Signal_heads_states(j,i,k)=0;
113                     end
114                 end
115             end
116         end
117
118         % Useful matrix
119         links_lanes_list={};
```

```
120         for i=1:size(lanes_table_sc,1)
121             links_lanes_list{i}=[lanes_table_sc{i,1},lanes_table_sc{i,2}];
122             Link_length(i)=links.ItemByKey(LinkID_list(i)).AttValue('Length2D'); %
                    Link length in m
123             Link_name{i}=links.ItemByKey(LinkID_list(i)).AttValue('Name'); %Link
                    length in m
124         end
125
126         % [Nominal Demand Matrix 1] x [Inflow]
127         for row=1:demand_matrix1.RowCount()
128             for col=1:demand_matrix1.ColCount()
129                 matrix_cell=demand_matrix1.GetValue(row,col)*inflow;
130                 demand_matrix1.SetValue(row,col,matrix_cell)
131             end
132         end
133
134         % [Nominal Demand Matrix 2] x [Inflow]
135         for row=1:demand_matrix2.RowCount()
136             for col=1:demand_matrix2.ColCount()
137                 matrix_cell=demand_matrix2.GetValue(row,col)*inflow;
138                 demand_matrix2.SetValue(row,col,matrix_cell)
139             end
140         end
141
142         % Filter Data
143         q_pred=zeros(size(lanes_table_sc,1),particles);
144         q_post=zeros(size(lanes_table_sc,1),particles);
145         QNF_prev=zeros(size(lanes_table_sc,1),seconds+sampling_time);
146
147         % DataCollector Measurements Data
148         DCMeas=vnet.DataCollectionMeasurements.GetAll;
149         DCM_Vehs=zeros(size(DCMeas,1),seconds/60);
150         DCM_Speed=zeros(size(DCMeas,1),seconds/60);
151
152         for dcm=1:size(DCMeas,1)
153             DCM_Name{dcm}=DCMeas{dcm}.AttValue('Name');
154         end
155         clear('dcm');
156
157         toc
158
159         %% SIMULATION
160         sim.set('AttValue','SimRes',1); % Resolution
161         eval=vissim_com.Evaluation;
162         set(vissim_com.Graphics,'AttValue','Quickmode',1); % Enables QuickMode
163
164         vehicles_data=zeros(vehs.count,2); % [delay,time in network] x vehicle
165
166         disp('Simulation launched') % Show Message
167
168         for k=1:seconds
169
170             vissim_com.Simulation.RunSingleStep; % Run next Step (1 second)
171
```

```
172             vehdeparted(k)=vehs.GetDeparted.Count; % Vehicles departed per step (
                    per second)
173             vehevolution(k)=vehs.count;
174             if k>1
175                 veharrived(k)=vehevolution(k-1)-vehevolution(k); % Vehicles arrived
                        per step (per second)
176             end
177
178             %-----------------------------------------------------------------
179             % Initialize Signal Controllers per k==1
180             if k==1
181                 message=strcat('Progress: ',num2str(k/seconds*100,4),'% [',num2str(
                        k),'s]');
182                 disp(message)
183                 for j=1:size(network_sc,1)
184                     for i=1:network_sc{j,2}
185                         set(sgs{j}.ItemByKey(i),'AttValue','ContrByCOM',1);
186                         set(sgs{j}.ItemByKey(i),'AttValue','State',1);
187                         if i==1
188                             set(sgs{j}.ItemByKey(i),'AttValue','State',3);
189                         end
190                     end
191                 end
192             end
193
194             %-----------------------------------------------------------------
195             % Save DataCollector Data
196             if mod(k,60)==0
197                 for dcm=1:size(DCMeas,1)
198                     DCM_Vehs(dcm,k/60)=DCMeas{dcm}.AttValue('Vehs(Current,Last,All)
                            ');
199                     DCM_Speed(dcm,k/60)=DCMeas{dcm}.AttValue('SpeedAvgArith(Current
                            ,Last,All)');
200                 end
201             end
202
203             %-----------------------------------------------------------------
204             % Save Screenshots
205             if k==seconds/2-1 || k==seconds-1
206                 set(vissim_com.Graphics,'AttValue','Quickmode',0);
207             end
208
209             if k==seconds/2 || k==seconds
210                 vissim_com.Graphics.CurrentNetworkWindow.Screenshot(strcat('
                        results_sims\screenshots\','case_',num2str(num_case),num2str(
                        noise*100),'_',num2str(inflow*100,'%03d'),'_',num2str(k),'s','.
                        jpg'),1.0)
211             end
212
213             if k==seconds/2+1
214                 set(vissim_com.Graphics,'AttValue','Quickmode',1);
215             end
216
217             %-----------------------------------------------------------------
```

```
218             % Show message
219             if mod(k,Tsig)==0
220                 fprintf(repmat('\b', 1, length(message)+1));
221                 message=strcat('Progress: ',num2str(k/seconds*100,4),'% [',num2str(
                        k),'s]');
222                 disp(message)
223             end
224 
225             %---------------------------------------------------------------------
226             % DATA COLLECTION
227             % Checks the vehs in every lane of every link used for the Back-
                    Pressure signaling control algorithm
228             if mod(k,sampling_time)==0
229 
230                 parfor i=1:size(lanes_table_sc,1)
231 
232                     % LinkID = lanes_table_sc{i,1};
233                     % LaneID = lanes_table_sc{i,2};
234 
235                     list_vehs_LaneID = links.ItemByKey(lanes_table_sc{i,1}).Lanes.
                            ItemByKey(lanes_table_sc{i,2}).Vehs.GetAll;
236 
237                     vehicles_data_temp{i,k}=[];
238                     if size(list_vehs_LaneID,1)>0
239                         vehicles_data_temp{i,k}=size(size(list_vehs_LaneID,1),3);
240                     end
241 
242                     for z=1:size(list_vehs_LaneID,1)
243 
244                         veh = list_vehs_LaneID{z};
245 
246                         veh_speed=veh.AttValue('Speed');
247 
248                         Volume_t(i,k)=Volume_t(i,k)+1;
249                         Avg_Speed(i,k)=Avg_Speed(i,k)+veh_speed;
250 
251                         dtm=veh.AttValue('DelayTm');
252                         tin=veh.AttValue('TmInNetTot');
253 
254                         vehicles_data_temp{i,k}(z,1)=veh.AttValue('No');
255                         vehicles_data_temp{i,k}(z,2)=dtm;
256                         vehicles_data_temp{i,k}(z,3)=tin;
257 
258                         delaytime(i,k)=delaytime(i,k)+dtm;
259                         timeinnetwork(i,k)=timeinnetwork(i,k)+tin;
260 
261                         if veh_speed< 4
262                             Q(i,k)=Q(i,k)+1;
263                         end
264 
265                         if veh_speed>4
266                             Speed(i,k)=Speed(i,k)+veh_speed;% Accumulated speed of
                                    cars in free cell
267                         end
```

```
268
269                 end
270
271                 % average speed in LaneID at k
272                 if Avg_Speed(i,k)~=0
273                     Avg_Speed(i,k)=Avg_Speed(i,k)/size(list_vehs_LaneID,1);
274                 else
275                     Avg_Speed(i,k)=0;
276                 end
277
278                 Volume_t(i,k)=size(list_vehs_LaneID,1);
279
280                 % Noise Level: avg length car = 5m
281                 % Number of maximum cars in a lane * % of error
282                 NL(i,k)=round(Link_length(i)/5*noise);
283                 % Queue with noise
284                 QN(i,k)=max(Q(i,k)+randi([-NL(i,k) NL(i,k)],1,1),0);
285
286                 % Prediction
287                 pd = makedist('Normal','mu',0,'sigma',4);
288                 tpd = truncate(pd,-5,5);
289                 tpdm = round(random(tpd,1,100));
290                 q_pred(i,:)=Q(i,k)+tpdm;
291
292                 % Run Particle Filter
293
294                 [QNF(i,k),q_post(i,:)] = particle_filter(QN(i,k),NL(i,k),q_pred
                        (i,:),particles);
295
296             end
297
298             QNF_prev(:,sampling_time*2:seconds+sampling_time)=QNF(:,
                    sampling_time:seconds);
299
300         end
301
302         %-----------------------------------------------------------------------
303         % BACK PRESSURE Algorithm
304
305         if mod(k,Tsig)==0 && k<seconds
306
307             parfor j=1:size(network_sc,1)
308
309                 % Pressure release calculation
310
311                 pa=zeros(1,network_sc{j,2});
312                 for phase_sc=1:network_sc{j,2}
313
314                     for option_sc=1:size(network_sc{j,((phase_sc-1)*4+4)}{:},1)
315                         queue_cars_sc_in=0;
316                         queue_cars_sc_out=0;
317                         for option_sc_2=1:size(network_sc{j,((phase_sc-1)*4+4)
                                }{:},2)   % IN Links and Lanes + queue
318
```

```
319                             link_sc_in = network_sc{j,((phase_sc-1)*4+4)}{:}(
                                    option_sc,option_sc_2);
320                             lane_sc_in = network_sc{j,((phase_sc-1)*4+5)}{:}(
                                    option_sc,option_sc_2);
321
322                             if link_sc_in ~= 0 || lane_sc_in ~= 0
323                                 queue_cars_sc_in = queue_cars_sc_in + QNF(
                                        conversion_list(link_sc_in,lane_sc_in),k);
324                             end
325                         end
326
327                         for option_sc_2=1:size(network_sc{j,((phase_sc-1)*4+6)
                                }{:},2)   % OUT Links and Lanes + queue
328
329                             link_sc_out = network_sc{j,((phase_sc-1)*4+6)}{:}(
                                    option_sc,option_sc_2);
330                             lane_sc_out = network_sc{j,((phase_sc-1)*4+7)}{:}(
                                    option_sc,option_sc_2);
331
332                             if link_sc_out ~= 0 || lane_sc_out ~= 0
333                                 queue_cars_sc_out = queue_cars_sc_out + QNF(
                                        conversion_list(link_sc_out,lane_sc_out),k)
                                        ;
334                             end
335                         end
336
337                         pa(phase_sc) = pa(phase_sc) + max( queue_cars_sc_in -
                                queue_cars_sc_out , 0 );  % PA calc
338
339                     end
340                 end
341
342                 [maxx,ind(j)]=max(pa);
343                 s=find(maxx-pa==0);
344                 if length(s)>1
345                     ind(j)=s(randi([1 length(s)],1));
346                 end
347
348                 % modify the signal heads
349                 if ind(j)~=prev_ind(j)
350                     set(sgs{j}.ItemByKey(ind(j)),'AttValue','State',3);
                                % Green Light
351                     set(sgs{j}.ItemByKey(prev_ind(j)),'AttValue','State',1);
                                % Red Light
352                     prev_ind(j)=ind(j);
353                 end
354
355             end       %end of junctions
356
357
358         end
359
360         % Record all the Signal Heads states
361         for j=1:size(ind,2)
```

```
                Signal_heads_states(j,ind(j),k)=3;
            end

        end

        %% SAVE RESULTS

        disp('End of Simulation')
        toc
        disp('Saving data...')

        for k=sampling_time:sampling_time:seconds
            for i=1:size(lanes_table_sc,1)
                for z=1:size(vehicles_data_temp{i,k},1)
                    vehicles_data(vehicles_data_temp{i,k}(z,1),1)=
                        vehicles_data_temp{i,k}(z,2);
                    vehicles_data(vehicles_data_temp{i,k}(z,1),2)=
                        vehicles_data_temp{i,k}(z,3);
                end
            end
        end

        output_file_name=strcat('results_sims\','data_case_',num2str(num_case),
            num2str(noise*100),'_',num2str(inflow*100,'%03d'),'.mat');
        save(output_file_name,'num_case','lanes_table_sc','network_sc','Avg_Speed',
            'Link_length','Link_name','Q','QN','QNF','Speed','Volume_t','
            Signal_heads_states','inflow','DCM_Name','DCM_Speed','DCM_Vehs','NL','
            prediction','delaytime','timeinnetwork','vehdeparted','vehicles_data','
            veharrived','vehevolution');

        toc
        fprintf('Data Saved\n\n');

    end
end
```

C.2 Particle filter function

```
function [q_est,qtmp] = particle_filter(QN,NL,q_pred,particles)

w=ones(1,particles);

shift=0:particles-1;
shift=shift'*particles;

for k=1:particles
    if (q_pred(k)<max(QN-NL,0))
        w(k)=0;
    end
    if (q_pred(k)>QN+NL)
        w(k)=0;
    end
```

```
end

if (sum(w)>0)
    w=w/sum(w);
    qtmp=q_pred(find(mnrnd(1,w,particles)')-shift);       % resample
else
    qtmp=q_pred;                                          % no resample
end

q_est=mode(qtmp);
end
```

C.3 Import Network data from .csv file function

```
function [output_data1,output_data2] = convert_data(file)
% example file = 'sc_data_kamppi5.csv'
% table=convert_data('sc_data_kamppi5.csv');

network_sc = readtable(file);

for row=1:size(network_sc,1)
%     disp(row)
    for column=4:(network_sc{row,2}*4+3)

        cell = network_sc{row,column}{:};
        cell = strrep(cell,'[','');
        cell = strrep(cell,']','');
        cell  = strsplit(cell,';');


        matrix=[];
        for i=1:size(cell,2)
            row_matrix=[];
            if network_sc{row,3}==1

                row_cell = strsplit(cell{i});
                for j=1:size(row_cell,2)
                    row_matrix = [ row_matrix str2double(row_cell(j)) ];
                end
                matrix = [ matrix ; row_matrix ];
            else

                matrix = [ matrix ; str2double(cell(i)) ];

            end

        end

        network_sc{row,column}{:} = matrix;

    end
end
```

```

% Table of Links and Lanes used for the SC (Where the queues take place)

lanes_table_sc=table();
test_table=table();
for row=1:size(network_sc,1)
    for column=1:(network_sc{row,2}*2)
        for i=1:size(network_sc{row,((column-1)*2+4)}{:},1)
            for j=1:size(network_sc{row,((column-1)*2+5)}{:},2)
                test_table.Link = network_sc{row,((column-1)*2+4)}{:}(i,j);
                test_table.Lane = network_sc{row,((column-1)*2+5)}{:}(i,j);
                lanes_table_sc=[lanes_table_sc;test_table];
            end
        end
    end
end

lanes_table_sc=unique(lanes_table_sc);
lanes_table_sc(1,:)=[];

save 'kamppi_network_sc' network_sc
save 'kamppi_lanes_table_sc' lanes_table_sc
output_data1 = network_sc;
output_data2 = lanes_table_sc;
end
```

C.4 Plot NFD - script

```
clear all;
clc;
close all;

%% Directory

data_directory='results_sims';
output_directory='figures';
mkdir(output_directory);

if exist(strcat(output_directory,'\','NFD\'),'dir')==0
    mkdir(strcat(output_directory,'\','NFD\'));
end

%% NFD

% Parameters
num_case=425;
inflow_values=2.0;
seconds=3600;
sampling_time=5;

figure_1=figure('Name','Flow vs Density','NumberTitle','off');
% lgd=legend;
```

```
% lgd.Location='south';
hold on
for inflow=1:size(inflow_values,1)

    input_file_name=strcat(data_directory,'\','data_case_',num2str(num_case),'_',
        num2str(inflow_values(inflow)*100,'%03d'),'.mat');
    load(input_file_name,'vehdeparted','veharrived')

    vehinnetwork=cumsum(vehdeparted)-cumsum(veharrived);

    veharrivedtemp=0;
    for i=1:seconds
            veharrivedtemp=veharrivedtemp+veharrived(i);
        if mod(i,60)==0
            veharrived60(i/60)=veharrivedtemp;
            veharrivedtemp=0;
            vehinnetwork60(i/60)=vehinnetwork(i);
            veharrivedtemp=0;
        end
    end


    plot(vehinnetwork60,veharrived60,'LineStyle','none','Marker','.','DisplayName',
        num2str(inflow_values(inflow)*100,'%03d'));

end

hold off
xlabel('Vehicles in the network (Density) [Vehicles]')
ylabel('Flow [Vehicles/min]')
title('Flow vs Density');
axis([0 inf 0 inf])
saveas(gcf,strcat(output_directory,'\NFD\',num2str(num_case),'_',num2str(
    inflow_values(inflow)*100,'%03d'),'_','NFD'),'epsc');
saveas(gcf,strcat(output_directory,'\NFD\',num2str(num_case),'_',num2str(
    inflow_values(inflow)*100,'%03d'),'_','NFD'),'jpg');
```

C.5 Plot Overall Results - script

```
clear all;
clc;
close all;

%% Directory

data_directory='results_sims';
output_directory='figures';
if exist(output_directory,'dir')==0
    mkdir(output_directory);
end
if exist(strcat(output_directory,'\','general\'),'dir')==0
    mkdir(strcat(output_directory,'\','general\'));
```

```
14  end
15
16  %% Important data
17  inflow_values=[0.5,0.75,1.0,1.25,1.5,2.0,2.5];
18  cases_values=[100,200,310,325,350,410,425,450];
19  sampling_time=5;
20  seconds=3600;
21
22  total_cases=size(inflow_values,1);
23
24
25  %% Figures
26  for inflow=inflow_values
27
28      Q_cases=cell(total_cases,1);
29      Q_sum=zeros(total_cases,seconds/sampling_time);
30
31      for num_case=cases_values
32
33          input_file_name=strcat(data_directory,'\','data_case_',num2str(num_case),'_
                ',num2str(inflow*100,'%03d'),'.mat');
34          load(input_file_name,'Q','Avg_Speed','vehdeparted','vehicles_data');
35
36          Q_temp=zeros(size(Q,1),seconds/sampling_time);
37
38          for i=1:seconds/sampling_time
39              for j=1:size(Q,1)
40                  Q_temp(j,i)=Q(j,i*sampling_time);
41                  Avg_Speed_temp(j,i)=Avg_Speed(j,i*sampling_time);
42              end
43          end
44
45          Q_cases{cases_values==num_case}=Q_temp;
46          Q_sum(cases_values==num_case,:)=sum(Q_temp);
47          Q_avg(cases_values==num_case,:)=sum(Q_temp)/size(Q,1);
48          Q_mean(cases_values==num_case,inflow_values==inflow)=mean(Q_avg(
                cases_values==num_case,:));
49          Q_std(cases_values==num_case,inflow_values==inflow)=std(Q_avg(cases_values
                ==num_case,:));
50
51          vehperhour(cases_values==num_case,inflow_values==inflow)=sum(vehdeparted);
52
53          traveltime_mean(cases_values==num_case,inflow_values==inflow)=sum(
                vehicles_data(:,2))/size(find(vehicles_data(:,2)),1);
54
55          Avg_Speed_mean(cases_values==num_case,inflow_values==inflow)=mean(sum(
                Avg_Speed_temp)/size(Avg_Speed,1));
56
57          clear('Q','Avg_Speed','vehdeparted','vehicles_data');
58
59      end
60
61  end
62
```

```
name_noises={'10','25','50'};
labels={'0.5','0.75','1.0','1.25','1.5','2.0','2.5'};
for index=[0,1,2]
    name_fig='Avg_Q_length';
    name_case_3=strcat('case 3',name_noises{index+1},': BP Control, ',name_noises{
        index+1},'% noise');
    name_case_4=strcat('case 4',name_noises{index+1},': BP Control, ',name_noises{
        index+1},'% noise, filtered');
    figure_1=figure('Position', [50 50 640 480]);
    hold on
    plot(vehperhour(1,:),Q_mean(1,:),'Marker','.');
    text(vehperhour(1,:),Q_mean(1,:),labels,'VerticalAlignment','bottom','
        HorizontalAlignment','right','FontSize',6)
    lgd=legend('case 100: Fixed and cyclic Signal Heads');
    lgd.Location='northwest';
    plot(vehperhour(2,:),Q_mean(2,:),'Marker','.','DisplayName','case 200: BP
        Control without noise');
    plot(vehperhour(3+index,:),Q_mean(3+index,:),'Marker','.','DisplayName',
        name_case_3);
    plot(vehperhour(6+index,:),Q_mean(6+index,:),'Marker','.','DisplayName',
        name_case_4);
    hold off
    xlabel('Inflow [vehicles/hour]')
    ylabel('Avg Queue length [vehicles]')

    saveas(gcf,strcat(output_directory,'\general\',name_fig,'_N',name_noises{index
        +1}),'epsc');
    saveas(gcf,strcat(output_directory,'\general\',name_fig,'_N',name_noises{index
        +1}),'jpg');
    close all;

    name_fig='Avg_Speed';
    name_case_3=strcat('case 3',name_noises{index+1},': BP Control, ',name_noises{
        index+1},'% noise');
    name_case_4=strcat('case 4',name_noises{index+1},': BP Control, ',name_noises{
        index+1},'% noise, filtered');
    figure_1=figure('Position', [50 50 640 480]);
    hold on
    plot(vehperhour(1,:),Avg_Speed_mean(1,:),'Marker','.');
    text(vehperhour(1,:),Avg_Speed_mean(1,:),labels,'VerticalAlignment','bottom','
        HorizontalAlignment','right','FontSize',6)
    lgd=legend('case 100: Fixed and cyclic Signal Heads');
    lgd.Location='southwest';
    plot(vehperhour(2,:),Avg_Speed_mean(2,:),'Marker','.','DisplayName','case 200:
        BP Control without noise');
    plot(vehperhour(3+index,:),Avg_Speed_mean(3+index,:),'Marker','.','DisplayName'
        ,name_case_3);
    plot(vehperhour(6+index,:),Avg_Speed_mean(6+index,:),'Marker','.','DisplayName'
        ,name_case_4);
    hold off
    xlabel('Inflow [vehicles/hour]')
    ylabel('Avg Speed [km/hour]')

```

```
102     saveas(gcf,strcat(output_directory,'\general\',name_fig,'_N',name_noises{index
            +1}),'epsc');
103     saveas(gcf,strcat(output_directory,'\general\',name_fig,'_N',name_noises{index
            +1}),'jpg');
104     close all;
105
106     name_fig='Avg_Travel_Time';
107     name_case_3=strcat('case 3',name_noises{index+1},': BP Control, ',name_noises{
            index+1},'% noise');
108     name_case_4=strcat('case 4',name_noises{index+1},': BP Control, ',name_noises{
            index+1},'% noise, filtered');
109     figure_1=figure('Position', [50 50 640 480]);
110     hold on
111     plot(vehperhour(1,:),traveltime_mean(1,:),'Marker','.');
112     text(vehperhour(1,:),traveltime_mean(1,:),labels,'VerticalAlignment','bottom','
            HorizontalAlignment','right','FontSize',6)
113     lgd=legend('case 100: Fixed and cyclic Signal Heads');
114     plot(vehperhour(2,:),traveltime_mean(2,:),'Marker','.','DisplayName','case 200:
             BP Control without noise');
115     plot(vehperhour(3+index,:),traveltime_mean(3+index,:),'Marker','.','DisplayName
            ',name_case_3);
116     plot(vehperhour(6+index,:),traveltime_mean(6+index,:),'Marker','.','DisplayName
            ',name_case_4);
117     hold off
118     xlabel('Inflow [vehicles/hour]')
119     ylabel('Avg Travel Time [seconds]')
120
121     saveas(gcf,strcat(output_directory,'\general\',name_fig,'_N',name_noises{index
            +1}),'epsc');
122     saveas(gcf,strcat(output_directory,'\general\',name_fig,'_N',name_noises{index
            +1}),'jpg');
123     close all;
124
125     name_fig='Veh_departed';
126     name_case_3=strcat('case 3',name_noises{index+1},': BP Control, ',name_noises{
            index+1},'% noise');
127     name_case_4=strcat('case 4',name_noises{index+1},': BP Control, ',name_noises{
            index+1},'% noise, filtered');
128     figure_1=figure('Position', [50 50 640 480]);
129     hold on
130     plot(inflow_values,vehperhour(1,:),'Marker','.');
131     lgd=legend('case 100: Fixed and cyclic Signal Heads');
132     lgd.Location='southeast';
133     plot(inflow_values,vehperhour(2,:),'Marker','.','DisplayName','case 200: BP
            Control without noise');
134     plot(inflow_values,vehperhour(3+index,:),'Marker','.','DisplayName',name_case_3
            );
135     plot(inflow_values,vehperhour(6+index,:),'Marker','.','DisplayName',name_case_4
            );
136     hold off
137     ylabel('Departed Vehicles [vehicles]')
138     xlabel('Inflow Multiplier case')
139
```

```
140     saveas(gcf,strcat(output_directory,'\general\',name_fig,'_N',name_noises{index
            +1}),'epsc');
141     saveas(gcf,strcat(output_directory,'\general\',name_fig,'_N',name_noises{index
            +1}),'jpg');
142     close all;
143 end
```

C.6 Plot Sum of Queues - script

```
1  clear all;
2  clc;
3  close all;
4
5  %% Directory
6
7  data_directory='results_sims';
8  output_directory='figures';
9  if exist(output_directory,'dir')==0
10     mkdir(output_directory);
11 end
12 if exist(strcat(output_directory,'\','sum_queues\'),'dir')==0
13     mkdir(strcat(output_directory,'\','sum_queues\'));
14 end
15
16 %% Important data
17 inflow_values=[0.5,0.75,1.0,1.25,1.5,2.0,2.5];
18 cases_values=[100,200,310,325,350,410,425,450];
19 sampling_time=5;
20 seconds=3600;
21
22 total_cases=size(inflow_values,1);
23
24 %% Figures
25 for inflow=inflow_values
26 % for inflow=1:1
27
28     Q_cases=cell(total_cases,1);
29     Q_sum=zeros(total_cases,seconds/sampling_time);
30
31     for num_case=cases_values
32
33         input_file_name=strcat(data_directory,'\','data_case_',num2str(num_case),'_
               ',num2str(inflow*100,'%03d'),'.mat');
34         load(input_file_name,'Q');
35
36         Q_temp=zeros(size(Q,1),seconds/sampling_time);
37
38         for i=1:seconds/sampling_time
39             for j=1:size(Q,1)
40                 Q_temp(j,i)=Q(j,i*sampling_time);
41             end
42         end
```

```
43
44          Q_cases{cases_values==num_case}=Q_temp;
45          Q_sum(cases_values==num_case,:)=sum(Q_temp);
46
47          clear('Q');
48
49      end
50
51
52      %% Sum Queue cases: All 10%
53
54      name_fig='All_N10';
55      figure_1=figure('Position', [10 10 1024 768]);
56      hold on
57      plot(sampling_time:sampling_time:seconds,Q_sum(1,:));
58      lgd=legend('case 100: Fixed and cyclic Signal Heads');
59      lgd.Location='northwest';
60      plot(sampling_time:sampling_time:seconds,Q_sum(2,:),'DisplayName','case 200: BP
            Control without noise');
61      plot(sampling_time:sampling_time:seconds,Q_sum(3,:),'DisplayName','case 310: BP
            Control, 10% noise');
62      plot(sampling_time:sampling_time:seconds,Q_sum(6,:),'DisplayName','case 410: BP
            Control, 10% noise, filtered');
63      hold off
64      xlabel('Time [s]');
65      ylabel('Sum of Queues [Vehicles]');
66      title('Aggregated queue length  for Kamppi network');
67
68      saveas(gcf,strcat(output_directory,'\sum_queues\cases_',name_fig,'_SoQ_inf',
            num2str(inflow*100,'%03d')),'epsc');
69      saveas(gcf,strcat(output_directory,'\sum_queues\cases_',name_fig,'_SoQ_inf',
            num2str(inflow*100,'%03d')),'jpg');
70      close all;
71
72      %% Sum Queue cases: All
73
74      name_fig='All_N25';
75      figure_1=figure('Position', [10 10 1024 768]);
76      hold on
77      plot(sampling_time:sampling_time:seconds,Q_sum(1,:));
78      lgd=legend('case 100: Fixed and cyclic Signal Heads');
79      lgd.Location='northwest';
80      plot(sampling_time:sampling_time:seconds,Q_sum(2,:),'DisplayName','case 200: BP
            Control without noise');
81      plot(sampling_time:sampling_time:seconds,Q_sum(4,:),'DisplayName','case 325: BP
            Control, 25% noise');
82      plot(sampling_time:sampling_time:seconds,Q_sum(7,:),'DisplayName','case 425: BP
            Control, 25% noise, filtered');
83      hold off
84      xlabel('Time [s]');
85      ylabel('Sum of Queues [Vehicles]');
86      title('Aggregated queue length  for Kamppi network');
87
```

```
88      saveas(gcf,strcat(output_directory,'\sum_queues\cases_',name_fig,'_SoQ_inf',
            num2str(inflow*100,'%03d')),'epsc');
89      saveas(gcf,strcat(output_directory,'\sum_queues\cases_',name_fig,'_SoQ_inf',
            num2str(inflow*100,'%03d')),'jpg');
90      close all;
91
92      %% Sum Queue cases: All
93
94      name_fig='All_N50';
95      figure_1=figure('Position', [10 10 1024 768]);
96      hold on
97      plot(sampling_time:sampling_time:seconds,Q_sum(1,:));
98      lgd=legend('case 100: Fixed and cyclic Signal Heads');
99      lgd.Location='northwest';
100     plot(sampling_time:sampling_time:seconds,Q_sum(2,:),'DisplayName','case 200: BP
            Control without noise');
101     plot(sampling_time:sampling_time:seconds,Q_sum(5,:),'DisplayName','case 350: BP
            Control, 50% noise');
102     plot(sampling_time:sampling_time:seconds,Q_sum(8,:),'DisplayName','case 450: BP
            Control, 50% noise, filtered');
103     hold off
104     xlabel('Time [s]');
105     ylabel('Sum of Queues [Vehicles]');
106     title('Aggregated queue length  for Kamppi network');
107
108     saveas(gcf,strcat(output_directory,'\sum_queues\cases_',name_fig,'_SoQ_inf',
            num2str(inflow*100,'%03d')),'epsc');
109     saveas(gcf,strcat(output_directory,'\sum_queues\cases_',name_fig,'_SoQ_inf',
            num2str(inflow*100,'%03d')),'jpg');
110     close all;
111
112
113 end
```

C.7 Plot Mean and Standard Deviation - script

```
1  clear all;
2  clc;
3  close all;
4
5  %% Directory
6
7  data_directory='results_sims';
8  output_directory='figures';
9  if exist(output_directory,'dir')==0
10     mkdir(output_directory);
11 end
12 if exist(strcat(output_directory,'\','normal_dist\'),'dir')==0
13     mkdir(strcat(output_directory,'\','normal_dist\'));
14 end
15
16 %% Important data
```

```
inflow_values=[0.5,0.75,1.0,1.25,1.5,2.0,2.5];
cases_values=[100,200,310,325,350,410,425,450];
sampling_time=5;
seconds=3600;

total_cases=size(inflow_values,1);

%% Figures
for inflow=inflow_values
% for inflow=4:4

    Q_cases=cell(total_cases,1);
    Q_sum=zeros(total_cases,seconds/sampling_time);

    for num_case=cases_values

        input_file_name=strcat(data_directory,'\','data_case_',num2str(num_case),'_
            ',num2str(inflow*100,'%03d'),'.mat');
        load(input_file_name,'Q');

        Q_temp=zeros(size(Q,1),seconds/sampling_time);

        for i=1:seconds/sampling_time
            for j=1:size(Q,1)
                Q_temp(j,i)=Q(j,i*sampling_time);
            end
        end

        Q_cases{cases_values==num_case}=Q_temp;
        Q_sum(cases_values==num_case,:)=sum(Q_temp);
        Q_avg(cases_values==num_case,:)=sum(Q_temp)/size(Q,1);
        Q_mean(cases_values==num_case)=mean(Q_avg(cases_values==num_case,:));
        Q_std(cases_values==num_case)=std(Q_avg(cases_values==num_case,:));

        clear('Q');

    end

    max_avg=max(Q_mean+Q_std*3.5);
    min_avg=min(Q_mean-Q_std*3.5);

    interval=0.01;

    if inflow>5
        interval=0.1;
    end

    x=[min_avg:interval:max_avg];
    for num_case=cases_values
        y{cases_values==num_case}=normpdf(x,Q_mean(cases_values==num_case),Q_std(
            cases_values==num_case));
    end

```

```
68      save(strcat(output_directory,'\normal_dist\data_cases_','_inf',num2str(inflow
            *100,'%03d'),'.mat'),'Q_mean','Q_std','Q_avg','Q_sum','Q_cases','x','y');
69
70      %-------------------------------------------------------------------------
71
72      %%
73
74      name_fig='All_N10';
75      figure_1=figure('Position', [10 10 1024 768]);
76      hold on
77      plot(x,y{1});
78      lgd=legend('case 100: Fixed and cyclic Signal Heads');
79      plot(x,y{2},'DisplayName','case 200: BP Control without noise');
80      plot(x,y{3},'DisplayName','case 310: BP Control, 10% noise');
81      plot(x,y{6},'DisplayName','case 410: BP Control, 10% noise, filtered');
82      hold off
83      xlabel('Queue mean [vehicles]')
84      title('Normal dist of queue length for Kamppi network');
85
86      saveas(gcf,strcat(output_directory,'\normal_dist\cases_',name_fig,'_ND_inf',
            num2str(inflow*100,'%03d')),'epsc');
87      saveas(gcf,strcat(output_directory,'\normal_dist\cases_',name_fig,'_ND_inf',
            num2str(inflow*100,'%03d')),'jpg');
88      close all;
89
90      %% Sum Queue cases: All
91
92      name_fig='All_N25';
93      figure_2=figure('Position', [10 10 1024 768]);
94      hold on
95      plot(x,y{1});
96      lgd=legend('case 100: Fixed and cyclic Signal Heads');
97      plot(x,y{2},'DisplayName','case 200: BP Control without noise');
98      plot(x,y{4},'DisplayName','case 310: BP Control, 25% noise');
99      plot(x,y{7},'DisplayName','case 410: BP Control, 25% noise, filtered');
100     hold off
101     xlabel('Queue mean [vehicles]')
102     title('Normal dist of queue length for Kamppi network');
103
104     saveas(gcf,strcat(output_directory,'\normal_dist\cases_',name_fig,'_ND_inf',
            num2str(inflow*100,'%03d')),'epsc');
105     saveas(gcf,strcat(output_directory,'\normal_dist\cases_',name_fig,'_ND_inf',
            num2str(inflow*100,'%03d')),'jpg');
106     close all;
107
108     %% Sum Queue cases: All
109
110     name_fig='All_N50';
111     figure_1=figure('Position', [10 10 1024 768]);
112     hold on
113     hold on
114     plot(x,y{1});
115     lgd=legend('case 100: Fixed and cyclic Signal Heads');
116     plot(x,y{2},'DisplayName','case 200: BP Control without noise');
```

```
117     plot(x,y{5},'DisplayName','case 310: BP Control, 50% noise');
118     plot(x,y{8},'DisplayName','case 410: BP Control, 50% noise, filtered');
119     hold off
120     xlabel('Queue mean [vehicles]')
121     title('Normal dist of queue length for Kamppi network');
122 
123     saveas(gcf,strcat(output_directory,'\normal_dist\cases_',name_fig,'_ND_inf',
            num2str(inflow*100,'%03d')),'epsc');
124     saveas(gcf,strcat(output_directory,'\normal_dist\cases_',name_fig,'_ND_inf',
            num2str(inflow*100,'%03d')),'jpg');
125     close all;
126 
127 
128 end
```

D Implementation Guide

1. Previous Steps

1a. Required files

- kamppi_case_#.m (example: kamppi_case_4.m)
- kamppi_network_sc.mat
- kamppi_lanes_table_sc.mat
- particle_filter.m
- PTV Vissim files (.inpx, etc)

Important note:

kamppi_case_4.m corresponds to cases 410 and 425. The other main simulation files, *kamppi_case_1.m*, *kamppi_case_2.m* and *kamppi_case_3.m* are based in *kamppi_case_4.m*. By commenting parts of the *kamppi_case_4.m* main script and by changing the queue measurement used in the algorithm, the other simulation files can be obtained.

List of main simulation files:

- kamppi_case_1.m : case 100
- kamppi_case_2.m : case 200
- kamppi_case_3.m : cases 310 and 325
- kamppi_case_4.m : cases 410 and 425

1b. Network data

If new *kamppi_network_sc.mat* and *kamppi_network_sc.mat* need to be generated because the network has changed (links used for the controllers have changed, for example), the *convert_data.m* (see appendix C) function has to be executed. The function converts a in *.csv* format table, as seen in appendix B, into these two files.

2. Launch the simulation

2a. Adjust parameters

1. Open the *kamppi_case_#.m* file.
2. Make sure that *num_case* variable corresponds to the selected case #.
3. Adjust the inflow values multiplier that will be simulated, by changing $inflow\_values$.
4. Adjust the noise level values that will be simulated, by changing *noise_values*.

Optional:

- $Tsig$: control period time in seconds (phase duration).
- *seconds*: simulation time in seconds.
- *sampling_time*: sampling period of the data, in seconds.

2b. Start the simulation

Once the main file is ready, the code can be executed (MATLAB RUN BUTTON). All the combinations of the $inflow\_values$ and *noise_values* will be simulated, one after another. The simulations can be supervised thanks to the shell, where the progression will be displayed.

2c. Saved data

After each simulation, data is saved automatically in the *results_sims* folder. Screenshots of the network state at 50% and 100% of each simulation are also saved.

3. Plot results

3a. Required files

When all the simulations are completed, figures can be created. The required files are all the data that is wanted to be displayed, found in the *results_sims* folder.

Scripts:

- $plot\_nfd.m$: plots the NFD of the network.
- *plot_general.m*: displays all the data collected about the total queue, average speed, travel time and departed vehicles, depending on vehicle inflow.
- *plot_queues.m*: plots the sum of queues of the network per simulation time of specific cases.
- *plot_avg_queues.m*: plots the mean and the standard deviation of queues per simulation time of specific cases.

3b. Adjust parameters

List of parameters to be adjusted (it depends of the acquired data):

- $inflow\_values$: simulated inflow values.
- *noise_values*: simulated noise values.
- *seconds*: simulated time in seconds.
- *sampling_time*: sampling period used during the simulation, in seconds.

It is possible that more changes in the code have to be performed.

3c. Saved figures

Launch the desired scripts. All the generated figures are saved in the *figures* folder.